

很多,并且会激发学生的学习兴趣。

4. 自学性

一本书如果适合自学学习,对其语言要求比较高。写作风格不能枯燥无味,让人看一眼就拒人千里之外,而应该是风趣、幽默,重要知识点多举实际应用的案例,说明它们在实际生活中的应用,应该有画龙点睛的说明和知识背景介绍,对其应用需要注意哪些问题等都要有提示。

一书在手,从第一页开始的起点到最后一页的终点,如何使读者能快乐地阅读下去并获得知识?这是非常重要的问题。在数学上,两点之间的最短距离是直线。但在知识的传播中,使读者感到“阻力最小”的书才是好书。如同自然界中没有直流的河流一样,河水在重力的作用下一定沿着阻力最小的路径向前进。知识的传播与此相同,最有效的传播方式是传播起来损耗最小,阅读起来没有阻力。

欢迎联系清华大学出版社白立军老师投稿: bailj@tup.tsinghua.edu.cn。

2014年12月15日

12.2.3	界面设计和功能实现.....	360
12.2.4	定时推送状态栏提醒.....	377
12.2.5	菜单设计.....	379
12.3	本章小结.....	382
习题	382
参考文献	384

实例;若存在,则会把任务列表中在其之上的其他 Activity 取消并调用它的 `onNewIntent()` 方法。SingleInstance 模式只有一个实例,不允许有别的 Activity 存在,也就是说,一个实例堆栈中只有一个 Activity。

3.1.5 Context 及其在 Activity 中的应用

Context 的中文解释是“上下文”或“环境”,在 Android 中应该理解为“场景”。例如,正在打电话时,场景就是用户所能看到的在手机里显示出来的拨号键盘,以及虽然看不到,但是却在系统后台运行的对应着拨号功能的处理程序。

Context 描述的是一个应用程序环境的上下文信息,是访问全局信息(如字符串资源、图片资源等)的接口。通过它可以获取应用程序的资源 and 类,也包括一些应用级别操作,如启动 Activity、发送广播、接收 Intent 信息等。也就是说,如果需要访问全局信息,就要使用 Context。在代码段 3-5 中, this 指的是这个语句所在的 Activity 对象,同时也是这个 Activity 的 Context。

代码段 3-5 通过 Context 获取 Activity 上下文中的字符串信息

```
public class MainActivity extends ActionBarActivity {  
    private TextView tv;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        tv = new TextView(this);                //得到当前 Activity 的上下文信息  
        tv.setText(R.string.hello_world);        //通过 Context 得到字符串资源  
        setContentView(tv);  
    }  
}
```

Android 系统中有很多 Context 对象,例如前述的 Activity 继承自 Context,也就是说每一个 Activity 对应一个 Context。Service 也继承自 Context,每一个 Service 也对应一个 Context。

常用的 Context 对象有两种,一种是 Activity 的 Context,另一种是 Application 的 Context。二者的生存周期不同。Activity 的 Context 生命周期仅在 Activity 存在时,也就是说,如果 Activity 已经被系统回收了,那么对应的 Context 也就不存在了;而 Application 的 Context 生命周期却很长,只要应用程序运行着,这个 Context 就是存在的,所以要根据自己程序的需要使用合适的 Context。

3.2 布局文件及其加载

Activity 主要用于呈现用户界面,包括显示 UI 控件、监听并处理用户的界面事件并做出响应等。

代码段 3-18 绝对布局实现简易浏览器界面

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width "match parent"
    android:layout_height "match parent"
    android:padding="5dp">
    <EditText
        android:layout_x="5dp"
        android:layout_y="10dp"
        android:layout_width="180dp"
        android:layout_height="wrap_content"
        android:singleLine="true"
        android:text="http://www.baidu.com/" />
    <Button
        android:id="@+id/btn_back"
        android:text="后退"
        android:layout_x="185dp"
        android:layout_y="5dp"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content" />
    <Button
        android:id="@+id/btn_go"
        android:text="前往"
        android:layout_x="270dp"
        android:layout_y="5dp"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content" />
    <WebView
        android:id="@+id/webView"
        android:layout_x="5dp"
        android:layout_y="55dp"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1" />
</AbsoluteLayout>
```

3.4.3 相对布局 RelativeLayout

在相对布局中,子元素的位置是相对于兄弟元素或父容器而确定的,例如在某一个给定 View 对象的左边或者下面,或相对于某个特定区域的位置(如底部对齐、中间偏左)等来定位元素。在设计相对布局时,要按照元素之间的依赖关系排列,如 View A 的位置相对于 View B 来决定,则需要保证在布局文件中 View B 在 View A 的前面。还需要注意的是,在进行相对布局时要避免出现循环依赖,例如设置相对布局的父容器排列方式为

6.3 利用 Fragment 实现界面的切换

用 Activity 进行页面切换时,首先需要新建 Intent 对象,给该对象设置一些必要的参数,然后调用 startActivity()方法进行页面跳转。如果需要 Activity 返回结果,则调用 startActivityForResult()方法,在 onActivityResult()方法中获得返回结果。此外,每一个 Activity 都需要在 AndroidManifest.xml 文件中注册。

与 Activity 相比,Fragment 是更轻量级的控件,无须在 AndroidManifest.xml 文件中声明相关信息。在应用程序内部利用 Fragment 实现界面跳转比 Activity 更灵活,运行速度也更快。另外,由于 Fragment 可以动态地加载到 Activity 中,因此可以方便地实现屏幕上部分界面的切换。

Fragment 依赖于 Activity,其生命周期由宿主 Activity 通过 FragmentManager 和 FragmentTransaction 等相关的类进行管理。

【例 6 5】 示例工程 Demo_06_FragmentExchange 利用 Fragment 实现屏幕部分界面的切换。

实现步骤如下:

(1) 修改 activity_main.xml,在其中添加 FrameLayout 元素,作为 Fragment 对象的容器,代码如代码段 6-6 所示。

代码段 6-6 Activity 的布局文件 activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:text="示例:利用 Fragment 实现界面切换"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="16dp"
        android:textColor="#aa00ff"/>
    <RatingBar
        android:id="@+id/ratingBar1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        style="?android:attr/ratingBarStyleSmall"
        android:numStars="10" />
    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```

(2) 新建两个 Fragment 的布局文件: fragment_main.xml 和 fragment_new.xml, 代码如代码段 6-7 和代码段 6-8 所示。

代码段 6-7 第一个 Fragment 的布局文件 fragment_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:orientation="vertical">
    <TextView
        android:text="\n 这是一个 Fragment (MainFragment), 主界面!\n"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:textColor="#000000"
        android:layout_marginTop="@dimen/activity_vertical_margin"/>
    <Button
        android:id="@+id/btnGoNextFragment"
        android:text="切换到下一个界面"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

代码段 6-8 第二个 Fragment 的布局文件 fragment_new.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin">
    <TextView
        android:text="\n 这是一个新的 Fragment (NewFragment), 新界面!\n"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:textColor="#000000"
        android:layout_marginTop="@dimen/activity_vertical_margin"/>
    <TextView
        android:text="按返回键可回退到上一个界面"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="@dimen/activity_vertical_margin"/>
</LinearLayout>
```

(3) 新建两个 Fragment 的类文件, MainFragment.java 和 NewFragment.java, 分别重写其 onCreateView() 方法, 如代码段 6-9 和代码段 6-10 所示。

MainFragment.java 的代码中调用 addToBackStack(null) 方法的目的是将事务加入返回栈。这是为了支持回退键, 当用户按回退键, 就会返回上一个 Fragment。如果不将事务加入返回栈, 当用户按回退键时程序会退出。

代码段 6-9 MainFragment.java, 重写 onCreateView() 方法

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_main, container, false);
    rootView.findViewById(R.id.btnGoNextFragment).setOnClickListener(new
        View.OnClickListener() {
            @Override
            public void onClick(View view) {
                getFragmentManager().beginTransaction()
                    .replace(R.id.fragment_container, new NewFragment())
                    //R.id.fragment_container 是 Fragment 的容器
                    .addToBackStack(null)
                    //这样用户按回退键, 就会返回上一个 Fragment
                    .commit();
            }
        });
    return rootView;
}
```

代码段 6-10 NewFragment.java, 重写 onCreateView() 方法

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View root = inflater.inflate(R.layout.fragment_new, container, false);
    return root;
}
```

(4) 重写 MainActivity 的 onCreate() 方法, 动态加载第一个 Fragment, 这将是程序启动时加载的界面。代码如代码段 6-11 所示。

代码段 6-11 在 Activity 中动态加载第一个 Fragment

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    if (savedInstanceState == null) {
        getFragmentManager().beginTransaction()
            .add(R.id.container, new MainFragment())
    }
}
```

```
        .commit();  
    }  
}
```

在第一个 Fragment 中设置了一个按钮,点击这个按钮,就会加载第二个 Fragment。由于在加载第二个 Fragment 时调用了 `addToBackStack(null)` 方法,将事务加入返回栈,所以当用户按回退键时,就会返回第一个 Fragment。在加载第一个 Fragment 时,没有将事务加入返回栈,所以,当显示第一个 Fragment 时,如果用户按回退键则程序会退出。程序的运行结果如图 6-9 所示。在两个界面进行切换时,只更新 Fragment 部分的内容,界面上半部分是 Activity 中的内容,并不会随之更新。从这个示例也可以看出,利用 Fragment 可以方便地实现部分界面的切换。



图 6-9 利用 Fragment 实现界面的切换

6.4 利用 Fragment 实现侧滑菜单

侧滑菜单又称为侧边栏菜单、抽屉菜单,带有侧滑菜单的设计既可以解决手机屏幕空间不足的问题,又可以提升用户的交互体验。利用 Fragment 可以方便地实现带侧滑菜单的 Activity。

实现侧滑菜单需要使用 DrawerLayout 控件。DrawerLayout 是 Support Library 包中实现了侧滑菜单效果的控件。DrawerLayout 分为侧边菜单和主内容区两部分,侧边菜单可以根据手势展开与隐藏,主内容区的内容可以随着菜单的点击而变化。

使用 DrawerLayout 控件需要引入 `android.support.v4.jar` 这个包。如果找不到这个类,首先用 SDK Manager 工具更新 Android Support Library,然后在 `Android SDK\extras\android\support\v4` 路径下找到 `android.support.v4.jar`,复制到项目的 `libs` 路径,执行 `Add to Build Path` 即可。程序中需要导入 `android.support.v4.widget.DrawerLayout` 包。

6.4.1 主视图的布局

主 Activity 使用的界面布局中必须将 `<android.support.v4.widget.DrawerLayout>` 元

素作为布局的根元素,如代码段 6-12 所示。一般情况下,在 DrawerLayout 布局中只有两个子布局,一个是内容布局,另一个是侧滑菜单布局。代码段 6-12 中的<FrameLayout>元素是内容布局,这是一个 Fragment 的容器,用于显示程序的主视图界面; <ListView> 元素是侧滑菜单布局,用于显示侧滑菜单项列表。

代码段 6-12 主 Activity 的布局文件 activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <FrameLayout
        android:id="@+id/mainContainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
    <ListView
        android:id="@+id/navigation_drawer_list"
        android:layout_width="150dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"/>
</android.support.v4.widget.DrawerLayout>
```

创建主布局文件时需要注意以下几点:

(1) 主视图的宽高设置必须是 match_parent,这样当侧滑菜单隐藏时,主视图全部铺满 Activity。

(2) 主内容区的布局代码要放在侧滑菜单布局的前面,这样 DrawerLayout 才能正确判断谁是侧滑菜单,谁是主内容区。

(3) 必须显式指定侧滑菜单视图的 android:layout_gravity 属性。其值设置为 start,则从左向右滑出菜单,end 为从右向左滑出菜单。虽然属性值设为 left 和 right 也能实现此功能,但是 Google 不推荐使用 left 和 right。

(4) 侧滑菜单的宽度最好不要超过主屏幕宽度的一半,这样当菜单滑出时还能看到主视图。

6.4.2 侧滑菜单的布局和菜单事件的响应

侧滑菜单通常使用一个 ListView 列出导航菜单项,其内容需要 Adapter 来初始化,其初始化可以在 Activity 的 onCreate()方法中完成,如代码段 6-13 所示。

代码段 6-13 主 Activity 的布局文件 activity_main.xml

```
@Override
protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);  
setContentView(R.layout.activity_main);  
myDrawerListView= (ListView) findViewById(R.id.navigation_drawer_list);  
myDrawerListView.setAdapter(new ArrayAdapter<String> (  
    this,R.layout.list_item,new String[]{"spring","summer","autumn","winter" }));  
}
```

对于侧滑菜单事件的响应,可以由 `OnItemClickListener` 接口来监听,如代码段 6-14 所示。需要注意的是,每完成一次菜单事件的响应,都要调用 `DrawerLayout` 对象的 `closeDrawer()` 方法,关闭侧滑菜单。

代码段 6-14 侧边栏菜单事件的监听和响应

```
myDrawerListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
    @Override  
    public void onItemClick(AdapterView<?>parent, View view, int position, long id) {  
        //每次点击,都在主视图中动态加载一个 Fragment  
        FragmentManager fragmentManager = getFragmentManager();  
        switch (position){  
            case 0:  
                fragmentManager.beginTransaction()  
                    .replace(R.id.mainContainer, new MainFragment01())  
                    .commit();  
                break;  
            case 1:  
                //其余代码类似,省略  
        }  
        myDrawerLayout.closeDrawer(myDrawerListView);  
    }  
});
```

【例 6 6】 示例工程 `Demo_06_NavigationDrawer` 演示了侧滑菜单的实现方法。

本例的布局文件包括 3 个: `activity_main.xml`、`list_item.xml`、`fragment_main.xml`, 分别是主 Activity 界面布局、侧滑菜单项的布局、响应菜单的 Fragment 的布局。Java 类文件包括 5 个: `MainActivity.java`、`MainFragment01.java`、`MainFragment02.java`、`MainFragment03.java`、`MainFragment04.java`, 分别是程序入口的 Activity 以及响应菜单时切换的 Fragment。

`activity_main.xml` 布局文件的内容如代码段 6-15 所示。其中,根元素为 `DrawerLayout`,用于实现侧滑菜单,<ListView> 元素的 `android:background` 属性设置为 `#88ddddd`,可以实现半透明的菜单背景,运行效果如图 6-10 所示。

代码段 6-15 主 Activity 的布局文件 activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <FrameLayout
        android:id="@+id/mainContainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
    <ListView
        android:id="@+id/navigation_drawer_list"
        android:layout_width="150dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:choiceMode="singleChoice"
        android:background="#88ddddd"
        android:divider="@android:color/transparent"
        android:dividerHeight="0dp"/>
</android.support.v4.widget.DrawerLayout>
```



图 6 10 侧滑菜单的运行效果

每次点击,都在主视图中动态加载一个 Fragment。每个 Fragment 中显示一幅图像,例如 MainActivity01 的代码如代码段 6-16 所示。

代码段 6-16 Fragment 的代码

```
public class MainActivity01 extends Fragment {
    public MainActivity01() {
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_main, container, false);
        ImageView imageView= (ImageView)rootView.findViewById(R.id.section_image);
        imageView.setImageResource(R.drawable.spring);
        return rootView;
    }
}
```

Fragment 的布局文件 fragment_main.xml 如代码段 6-17 所示。

代码段 6-17 Fragment 的布局

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/section_label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <ImageView
        android:id="@+id/section_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

侧滑菜单事件的响应,由 onItemClickListener 接口来监听,如代码段 6-18 所示。

代码段 6-18 侧边栏菜单事件的监听和响应

```
myDrawerListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?>parent, View view, int position, long id) {
        //每次点击,都在主视图中动态加载一个 Fragment
        FragmentManager fragmentManager = getFragmentManager();
        switch (position){
```

```
        case 0:
            fragmentManager.beginTransaction()
                .replace(R.id.mainContainer, new MainFragment01())
                .commit();
            break;
        case 1:
            fragmentManager.beginTransaction()
                .replace(R.id.mainContainer, new MainFragment02())
                .commit();
            break;
        case 2:
            fragmentManager.beginTransaction()
                .replace(R.id.mainContainer, new MainFragment03())
                .commit();
            break;
        case 3:
            fragmentManager.beginTransaction()
                .replace(R.id.mainContainer, new MainFragment04())
                .commit();
            break;
    }
    myDrawerLayout.closeDrawer(myDrawerListView);
}
});
```

6.4.3 使用 Android Studio 提供的模板实现侧滑菜单

Android Studio 开发环境为开发者提供了很多 Activity 模板,可以使用其中的 Navigation Drawer Activity 模板创建一个包含带侧边栏菜单的 Activity 的工程项目。下面通过一个示例介绍设计过程。

【例 6-7】 示例工程 Demo_06_NavigationDrawerActivity 演示了如何使用 Navigation Drawer Activity 模板创建一个带侧滑菜单的 Activity。

本例的布局文件包括 4 个: activity_main.xml、app_bar_main.xml、nav_header_main.xml 和 content_main.xml,分别是主 Activity 界面布局、侧滑菜单项的布局、侧滑菜单的标题栏布局和响应菜单项的主内容区的布局。主内容区的内容可以通过加载 Fragment 实现。

首先,新建一个工程项目。在创建的过程中,选择 Navigation Drawer Activity 模板,如图 6-11 所示。

工程项目中会包含一个自动生成的带侧滑菜单的 Activity,其运行效果如图 6-12 所示。图中自动生成的侧滑菜单项定义在 res\menu 文件夹中的 activity_main_drawer.



图 6-11 选择 Navigation Drawer Activity 模板

xml 文件中,内容如代码段 6-19 所示。

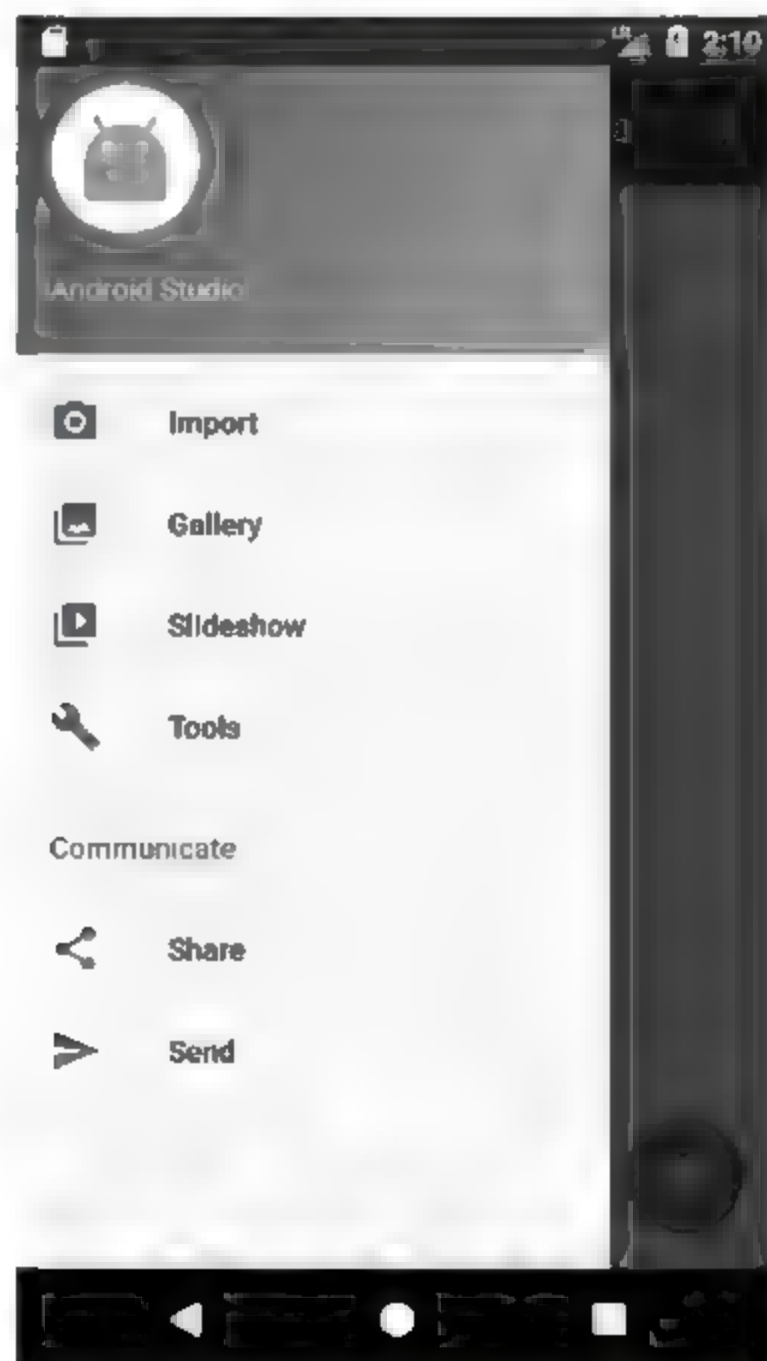


图 6 12 Activity 模板自动生成的侧滑菜单

代码段 6-19 activity_main_drawer.xml 的内容

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_camera"
            android:icon="@drawable/ic_menu_camera"
            android:title="Import"/>
        <item
            android:id="@+id/nav_gallery"
            android:icon="@drawable/ic_menu_gallery"
            android:title="Gallery"/>
        <item
            android:id="@+id/nav_slideshow"
            android:icon="@drawable/ic_menu_slideshow"
            android:title="Slideshow"/>
        <item
            android:id="@+id/nav_manage"
            android:icon="@drawable/ic_menu_manage"
            android:title="Tools"/>
    </group>
    <item android:title="Communicate">
        <menu>
            <item
                android:id="@+id/nav_share"
                android:icon="@drawable/ic_menu_share"
                android:title="Share"/>
            <item
                android:id="@+id/nav_send"
                android:icon="@drawable/ic_menu_send"
                android:title="Send"/>
        </menu>
    </item>
</menu>
```

系统监听到侧滑菜单动作后,会回调 MainActivity 中的 onNavigationItemSelected() 方法。该方法中已经包含了必要的框架代码,内容如代码段 6-20 所示,开发人员只需要按照自己的要求填充和改写即可。

代码段 6-20 onNavigationItemSelected() 方法的内容

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    //Handle navigation view item clicks here.
```

```
int id = item.getItemId();
if (id == R.id.nav_camera) {
    //处理对相机的操作
} else if (id == R.id.nav_gallery) {
} else if (id == R.id.nav_slideshow) {
} else if (id == R.id.nav_manage) {
} else if (id == R.id.nav_share) {
} else if (id == R.id.nav_send) {
}
DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
drawer.closeDrawer(GravityCompat.START);
return true;
}
```

6.5 利用 Fragment 实现 Tabbed Activity

除了 Navigation Drawer Activity 模板,Android Studio 还提供了 Tabbed Activity 模板,可以创建一个带有 3 个 Tab(标签页)的 Activity,实现多页面的切换。每一个标签页的内容都由 Fragment 呈现。

【例 6 8】 示例工程 Demo_06_TabbedActivity 演示了如何使用 Tabbed Activity 模板创建包含多个标签页的 Activity。

在工程中新建 3 个 Fragment 和相关的布局文件,用于显示每页想要显示的内容,方法与例 6-6 相同。

模板在 MainActivity.java 文件中已经定义了 SectionsPagerAdapter 类,这个类的 Fragment getItem(int position) 方法是响应标签页切换的回调方法。在这个方法中,可以使用 switch 语句,根据不同的 position 显示不同的 Fragment,如代码段 6-21 所示。

代码段 6-21 重写 Fragment getItem(int position) 方法

```
public Fragment getItem(int position) {
    switch (position) {
        case 0:
            return new MainFragment01();
        case 1:
            return new MainFragment02();
        case 2:
            return new MainFragment03();
    }
    return null;
}
```

如果在 Activity 中要设置 3 个以上的标签页,则不仅要在代码段 6-21 所示的

getItem()方法中添加对新增加的标签页的响应代码,还需要在 SectionsPagerAdapter 类的 getCount()方法中设置标签的数量。另外,在 getPageTitle()方法中可以设置标签上的文字。

示例程序的运行结果如图 6-13 所示,左右滑动屏幕可以进行标签页的切换。

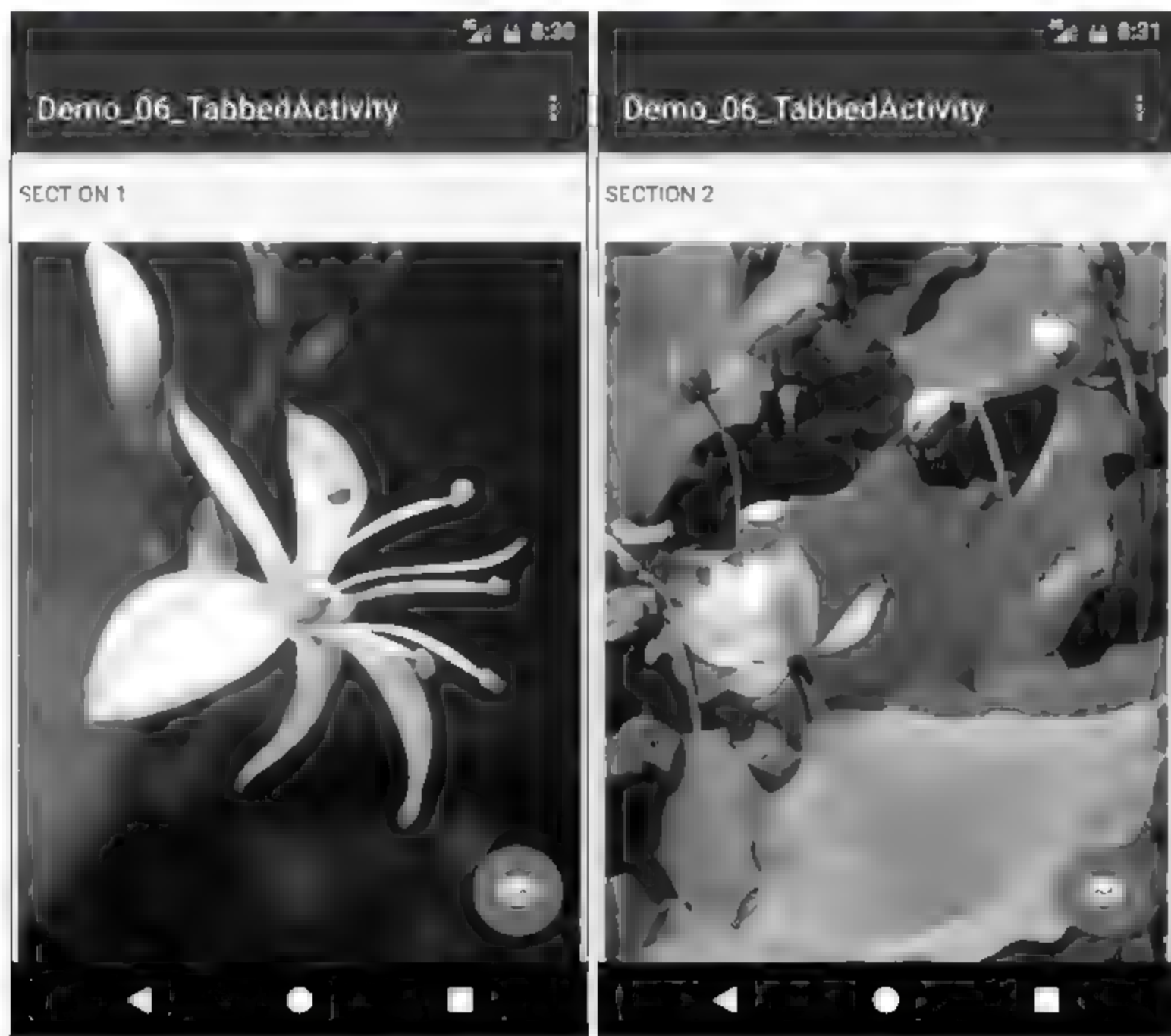


图 6-13 Tabbed Activity 的运行效果

6.6 本章小结

本章主要介绍了 Fragment 的概念和用法,并通过实例讲解了利用 Fragment 实现界面的切换、带侧滑菜单的 Activity、Tabbed Activity 的设计和实现方法。与 Activity 相比,Fragment 是一个轻量级控件,具有使用灵活、编程效率高、运行速度快等优点。学习本章时,要重点掌握如下内容:Fragment 的概念、用途和生命周期,利用 Fragment 实现界面切换的优点和方法,以及带侧边栏菜单 Activity 的设计和实现方法。

习 题

1. 编写一个程序,利用 System.out.println()验证 Fragment 的各生命周期方法满足什么条件时会被调用。请说明程序运行过程和验证结果。
2. 编写一个程序,利用 Fragment 实现屏幕部分界面的切换。
3. 利用 Fragment 设计一个注册的用户界面,注册项包括用户名、账号、密码、性别、

出生年月日、爱好。功能通过侧滑菜单实现,菜单项包括“清空各选项”“注册”“退出”。当用户点击“清空各选项”时,将所有文本输入框的文字清空,所有单选按钮和复选框设为启动时的默认选项;当用户点击“注册”时,显示状态栏提醒 Notification 消息,消息的标题为“注册完成”,消息中包括注册的用户名;当用户点击“退出”时,弹出警告对话框,用户如果选择“确定”,则关闭 Activity。

4. 设计一个应用程序,界面中有一个 TextView,其中显示一行文字。为应用程序添加侧滑菜单,包括“红”“绿”“蓝”3 个菜单项,用户选择一个菜单项,即将 TextView 中的文字设为相应的颜色。



Android 系统中同一进程里面的所有组件都是在 UI 线程中被实例化的,这个单线程模型可能会降低用户界面的响应速度,导致用户体验很差。在事件处理中使用多线程,将耗时处理过程转移到子线程上,就可以避免出现这种情况。本章介绍在 Android 系统中如何进行多线程操作,以及线程间通信的方法。在 Android 中有多种方法可以实现其他线程与 UI 线程通信,本章介绍常见的两种,即 Handler 和 AsyncTask。

7.1 基本概念

7.1.1 进程与线程

狭义的进程(process)是指正在运行的程序的实例。广义的进程是指计算机中的一个具有一定独立功能的程序关于某个数据集合的一次运行活动,是系统进行资源分配和调度的基本单位。在早期面向进程设计的计算机结构中,进程既是基本的分配单元,也是程序的基本执行单元;而在当代面向线程设计的计算机结构中,进程是线程的容器。

线程(thread)有时被称为轻量级进程(Lightweight Process, LWP),是程序执行流的最小单元。每一个程序都至少有一个线程,线程是进程中的一个实体,是被系统独立调度和分配的基本单位,线程与同属一个进程的其他线程共享进程所拥有的全部资源。一个线程可以创建和撤销另一个线程,同一进程中的多个线程之间可以并发执行。在单个程序中同时运行多个线程完成不同的工作,称为多线程。

在 Android 系统中,当一个应用程序第一次启动的时候,这个程序没有组件正在运行,系统会为这个程序以单一线程的形式启动一个新的 Linux 进程,这个线程一般称为程序的主线程(main thread)。默认的情况下,同一应用程序下的所有组件都将在该进程和线程中运行。同时,Android 会为每个应用程序分配一个单独的 Linux 用户。如果一个应用组件启动之前,这个应用的其他组件已经启动了,即这个应用的进程已经存在了,那么这个组件将会在这个进程中启动,同时在这个应用的主线程中执行。当然,也可以让应用中的组件运行在不同的进程中,也可以为任何进程添加额外的线程。

如果需要对程序中的某个组件运行在特定的进程中,可以在 AndroidManifest 文件中设置。AndroidManifest 文件中的<activity>、<service>、<receiver>和<provider>元素都有一个 process 属性来指定该组件运行在哪个进程中。可以设置成每个组件运行在

数据等。只要前台进程和可见进程有足够的内存,系统就不会回收它们。

(4) 后台进程(background process)。

后台进程运行着一个对用户不可见的 Activity,即调用过 onStop()方法的 Activity。这些进程对用户体验没有直接的影响,可以在服务进程、可见进程、前台进程需要内存的时候回收。通常,系统中会有很多不可见进程在运行,它们被保存在 LRU(Least Recently Used)列表中,以便内存不足的时候被第一时间回收。如果一个 Activity 正确地执行了它的生命周期回调方法,保存了自己的当前状态,关闭这个进程对于用户体验没有太大的影响。当用户回退到这个 Activity 时,它的所有可视状态将会被恢复。

(5) 空进程(empty process)。

空进程是一个不包含任何活动的应用组件的进程。运行这些进程的唯一原因是作为一个缓存,缩短下次程序需要重新使用的启动时间。系统经常中止这些进程,这样可以调节目程序缓存和系统缓存的平衡。

Android 是根据进程中组件的重要性尽可能高来评级的。例如,如果一个进程包含了一个 Service 和一个可见 Activity,那么这个进程将会被评为可见进程,而不是服务进程。

另外,当被另外的一个进程依赖的时候,某个进程的级别可能会增高。一个为其他进程服务的进程永远不会比被服务的进程重要性级别低。因为服务进程比后台 Activity 进程重要性级别高,因此一个要进行耗时工作的 Activity 最好启动一个 Service 来做这个工作,而不是开启一个子进程,特别是这个操作需要的时间比 Activity 存在的时间还要长的时候。例如,在后台播放音乐,向网络上传摄像头拍到的图片,使用 Service 可以使进程至少能获取到“服务进程”级别的重要性级别,而不用考虑 Activity 目前是什么状态。BroadcastReceiver 做费时工作的时候,也应该启用一个服务而不是开启一个线程。

7.1.2 创建线程

Java 提供了线程类 Thread 来创建多线程的程序,创建线程的操作与创建普通的类的对象是一样的,而线程就是 Thread 类或其子类的实例对象。每个 Thread 对象描述了一个单独的线程。

创建线程有两种方法。第一种方法是从 Java.lang.Thread 类派生一个新的线程类,通过构造方法来创建,如代码段 7-1 所示。

代码段 7-1 通过 Thread 类的构造方法创建线程

```
Thread thread = new Thread(new Runnable() {  
    @Override  
    public void run() {  
        //TODO Auto-generated method stub  
        //线程中要执行的操作  
    }  
});
```

第二种方法是通过实现 Runnable 接口来创建,实现 Runnable 接口中的 run() 方法,如代码段 7-2 所示。

代码段 7-2 实现 Runnable 接口创建线程

```
public class NewThread implements Runnable {  
    @Override  
    public void run() {  
        //TODO Auto-generated method stub  
        //线程中要执行的操作  
    }  
}
```

在 Java 中,由于类仅支持单继承,如果创建自定义线程类的时候是通过继承 Thread 类的方法来实现的,那么这个自定义类就不能再去继承其他的类,也就无法实现更加复杂的功能。因此,如果自定义类必须继承其他的类,那么就可以使用实现 Runnable 接口的方法来定义该类为线程类,这样就可以避免 Java 单继承所带来的局限性。实现 Runnable 接口相对于继承 Thread 类来说,不仅有利于程序的健壮性,使代码能够被多个线程共享,而且代码和数据资源相对独立,从而特别适合多个具有相同代码的线程处理同一资源的情况。这样线程、代码和数据资源三者有效分离,很好地体现了面向对象程序设计思想。

【例 7-1】 工程 Demo_07_NewThread 演示创建线程的方法。

示例程序中创建了一个计时器线程,并通过 LogCat 窗口输出计时结果。程序界面设置了两个按钮,分别用于暂停计时和继续计时。主要代码如代码段 7-3 所示。

代码段 7-3 多线程示例

```
public class MainActivity extends AppCompatActivity {  
    private String TAG = "线程示例";  
    private Button btnStart, btnEnd;  
    private Thread clockThread; //声明一个子线程,用于时钟计时  
    private boolean isRunning = false;  
    private int timer = 0;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        btnStart = (Button) findViewById(R.id.btnStart);  
        btnStart.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                isRunning = true;  
                clockThread.start(); //启动线程  
            }  
        });  
    }  
}
```

```
});  
btnEnd = (Button) findViewById(R.id.btnEnd);  
btnEnd.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        isRunning = false;  
    }  
});  
clockThread = new Thread(new Runnable() {  
    @Override  
    public void run() {  
        while (isRunning) {  
            try {  
                Thread.currentThread().sleep(1000);  
                timer++;  
                Log.d(TAG, "时间过去了: " + timer + " 秒");  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
});  
}
```

7.1.3 操作线程

不论以哪种方式创建线程,都必须调用 Thread 类中的 start() 方法来开启这个线程,也可以调用 sleep() 方法来让线程休眠指定的时间。当调用 start() 方法时线程开始工作,当线程中的 run() 方法执行完毕时,线程正常结束,也可以调用 interrupt() 或者 stop() 方法让线程结束。线程结束后,就无法重新启用了。

控制线程的常用方法如表 7-1 所示。

表 7-1 控制线程的常用方法

方法名称	功能及使用说明
public static void yield()	静态方法,将正在执行的线程放入到就绪队列中
public static void sleep(long millisec)	静态方法,在指定的毫秒数内让当前正在执行的线程休眠(暂停执行),此操作受到系统计时器和调度程序精度和准确性的影响
public static Thread currentThread()	静态方法,返回对当前正在执行的线程对象的引用
public void start()	使线程开始执行

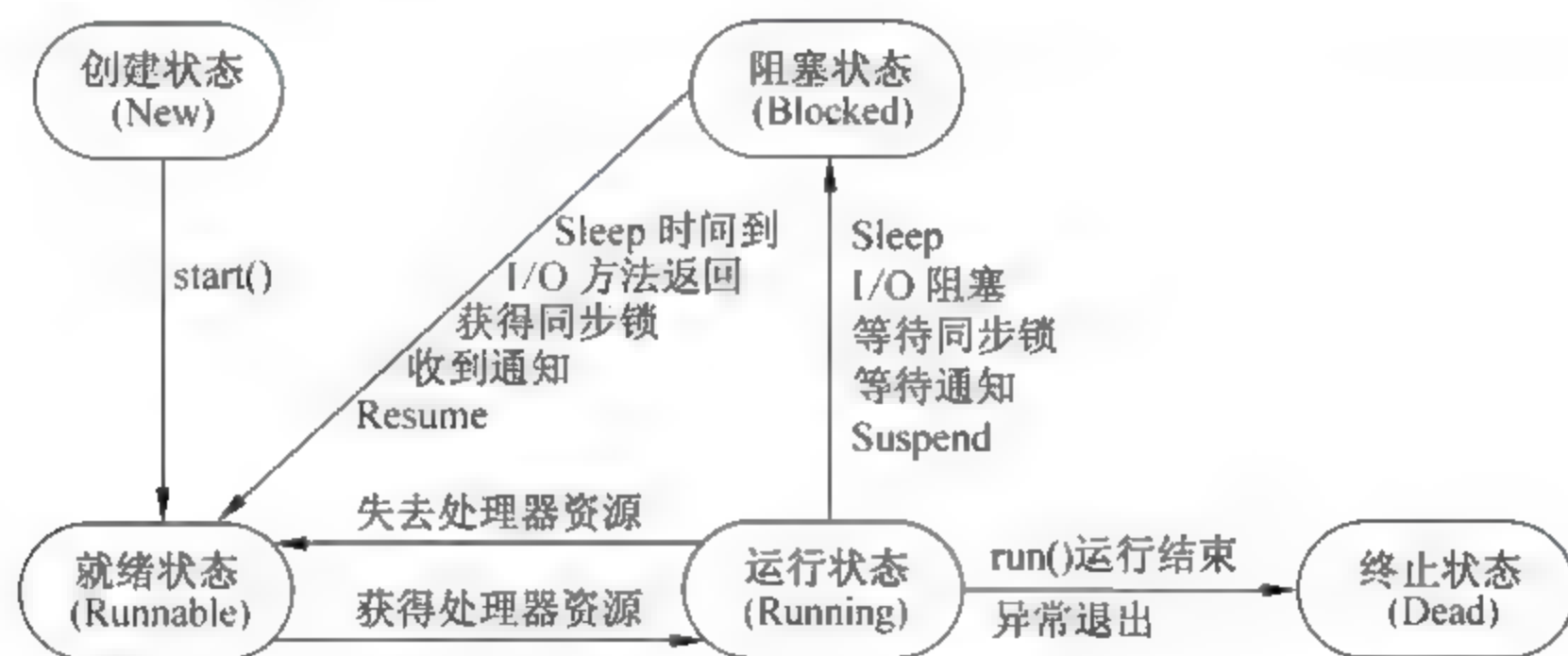
续表

方法名称	功能及使用说明
public void run()	由 Thread 实例调用,完成要执行的这个线程的内容
public final void setPriority(int priority)	更改线程的优先级
public final void setDaemon(boolean on)	将该线程标记为守护线程或用户线程
public final void join(long millsec)	等待该线程终止,参数为等待的最长时间。调用 join() 方法的线程执行完毕之后,才会执行下面的语句
public void interrupt()	中断线程
public final boolean isAlive()	测试线程是否处于活动状态

由于可能会引起死锁或不安全性,线程的很多方法现在已经不提倡使用了,例如 `destroy()`、`suspend()`、`resume()`、`stop()` 等,这里就不再介绍。

7.1.4 线程的状态和生命周期

一个线程从创建、启动到终止期间的任何时刻,总是处于以下 5 个状态中的某个状态,其状态图如图 7-1 所示。



(1) 创建状态(New)。

新创建了一个线程对象后,该线程对象就处于创建状态。处于创建状态的线程有自己的内存空间。

(2) 就绪状态(Runnable)。

线程对象创建后,其他线程调用了该对象的 `start()` 方法。该状态的线程位于可运行线程池中,变得可运行,等待获取 CPU 的使用权。注意,不能对已经启动的线程再次调用 `start()` 方法,否则会出现 `Java.lang.IllegalThreadStateException` 异常。

处于就绪状态的线程已经具备了运行条件,但还没有分配到 CPU,并不是运行状态,当系统选定一个等待执行的 Thread 对象后,它就会从等待执行状态进入执行状态,系统挑选的动作称为“CPU 调度”。一旦获得 CPU,线程就进入运行状态并调用自己的 `run()` 方法,执行 `run()` 方法中的任务。

通常,Activity 的生命周期方法,例如 onCreate()、onStart()、onResume()等,以及 Android 基类中以 on 开头的方法,例如 onClick()、onItemClick()等,都是在主线程被回调的,这意味着当系统调用这个组件时,这个组件不能长时间地阻塞主线程。例如,进行网络操作或更新 UI 时,如果运行时间较长,就不能直接在主线程中运行,应该将这样的组件分配到新建的线程中或是其他的线程中运行,避免负责界面更新的主线程无法处理界面事件,从而避免用户界面长时间失去响应。

总之,如果事件处理可能比较耗时,那么需要放到其他线程中处理,等处理完成后,再通知界面刷新,以保证应用程序良好的响应性。除此以外,有一些情况也适于使用多线程。例如,应用中有些情况下并不一定需要同步阻塞去等待返回结果,例如微博中的收藏功能,点击完收藏按钮后是否成功执行对当前的操作并没有影响,只需要完成后告诉用户就行了,这时可以通过多线程来实现异步。有时需要同时运行多任务,也可以使用多线程实现。

7.2.2 非 UI 线程访问 UI 对象

如前所述,为了避免阻塞 UI 线程,一些较费时的操作应该交给独立的子线程去执行。但是在开发 Android 应用时必须遵守单线程模型的原则,即 Android UI 操作并不是线程安全的,并且这些操作必须在 UI 线程中执行。也就是说,不能在一个子线程里访问 Android UI toolkit。如果子线程执行 UI 对象,Android 就会抛出异常 CalledFromWrongThreadException。

【例 7 2】 示例工程 Demo_07_WrongThreadUsing 演示了如何在一个子线程中计时并将计时的结果在一个 TextView 上显示。

本例修改了例 7 1 的程序,将 LogCat 输出语句改为调用 TextView 对象的 setText()方法,如代码段 7-4 所示。

代码段 7-4 在一个 worker 线程里访问 Android UI toolkit

```
clockThread = new Thread(new Runnable() {  
    @Override  
    public void run() {  
        while (isRunning) {  
            try {  
                Thread.currentThread().sleep(1000);  
                timer++;  
                tvTime.setText("时间过去了: " + timer + " 秒");  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
});
```

代码段 7-4 创建一个新的线程来做计时操作,但它违反了前述规则,在一个子线程修改了 UI 对象,即在非 UI 线程里访问了 Android UI toolkit。这会在程序执行过程中导致异常,如图 7-3 所示。

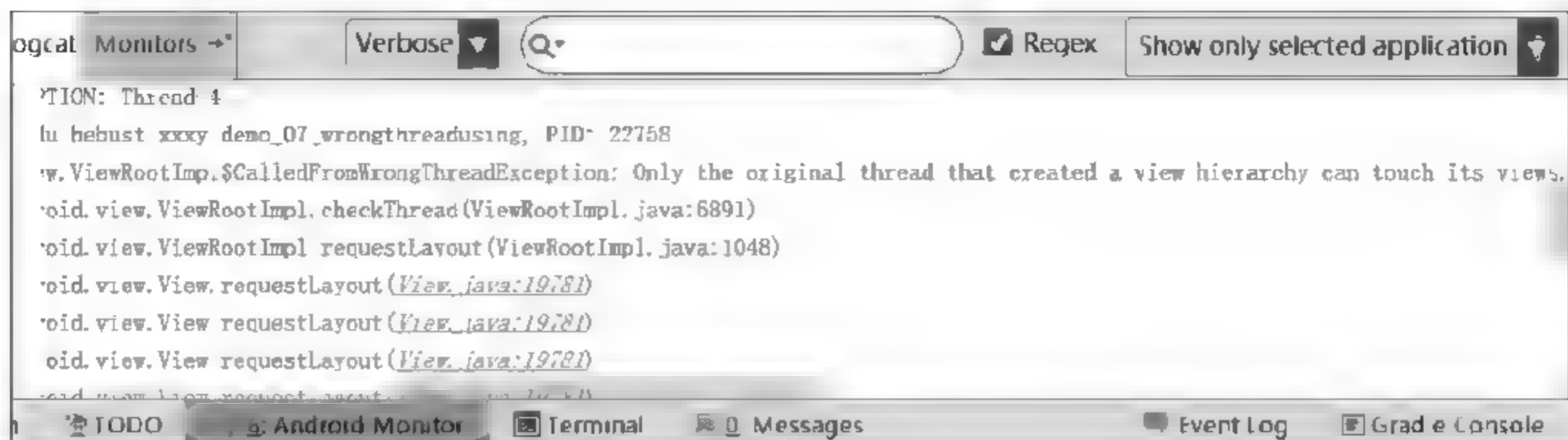


图 7-3 子线程修改 UI 对象导致异常

为了解决这个问题,Android 提供了如下几个方法从非 UI 线程访问 Android UI toolkit:

```
Activity.runOnUiThread(Runnable)
View.post(Runnable)
View.postDelayed(Runnable, long)
```

例如,可以使用 View.post(Runnable)方法来修改例 7 2 的代码,如代码段 7 5 所示。

代码段 7-5 使用 View.post(Runnable)方法访问 Android UI toolkit

```
clockThread = new Thread(new Runnable() {
    @Override
    public void run() {
        while (isRunning) {
            try {
                Thread.currentThread().sleep(1000);
                timer++;
                tvTime.post(new Runnable() {
                    @Override
                    public void run() {
                        tvTime.setText("时间过去了:" + timer + "秒");
                    }
                });
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
});
```

这个方案符合单线程模型的原则,计时操作在独立线程中完成,UI 线程对 TextView 进行操作。

然而,随着操作复杂性的增长,代码会变得越来越复杂,越来越难维护。为了用子线程处理更加复杂的交互,可以考虑在子线程中使用 Handler,用它来处理 UI 线程中的消息。更好的方法是继承 AsyncTask 类,这个类简化了需要同 UI 进行交互的子线程任务的执行。

另外,使用 `new Thread(){...}.start()` 这样的方式创建线程,如果在一个 Activity 中被多次调用,那么将创建多个匿名线程,程序运行得越久可能会越慢,使用一个 Handler 来启动、删除一个线程,可以保证线程不会重复地创建。

7.3 Android 多线程通信机制

把事件处理代码放到其他线程中处理,如果处理的结果需要刷新界面,那么需要线程间通信的方法来实现,在其他线程中发消息给 UI 线程处理。

Android 应用程序是通过消息来驱动的,系统为每一个应用程序维护一个消息队列,应用程序的主线程通过消息循环不断地从这个消息队列中获取消息,然后对这些消息进行处理,这样就实现了通过消息来驱动应用程序的执行。这样做的好处是:消息的发送方只需要把消息发送到主线程的消息队列中,而不需要等待消息的接收方处理完这个消息才返回,这样就可以提高系统的并发性。实质上,这就是一种异步处理机制。

在 Android 中有多种方法可以实现其他线程与 UI 线程通信,这里介绍常见的两种,即 Handler 和 AsyncTask。

7.3.1 线程间通信的常用类

Android SDK 提供了一系列类来管理线程以及线程间的通信。Android 的 Message 机制主要涉及 3 个主要的类,分别是 Handler、Looper、Message。

1. Handler 类

消息处理类 Handler 在 Android 系统里负责发送和处理消息,通过它可以实现其他线程与 UI 线程之间的消息通信。Handler 主要有两个用途:首先是可以定时处理或者分发消息,其次是可以添加一个执行的行为在其他线程中执行。

Handler 允许发送和处理 Message 或 Runnable 对象到其所在线程的 MessageQueue 中,在发送的时候可以指定不同的延迟时间、发送时间和要携带的数据。当 MessageQueue 循环到该 Message 时调用相应的 Handler 对象的 `handleMessage()` 方法对其进行处理。一个线程对应着一个 Looper 对象,一个 Looper 对象对应着一个 MessageQueue(消息队列)对象,但是一个线程可以有多个 Handler 对象,这些 Handler 对象可以共享同一个 Looper 和 MessageQueue。

Handler 的常用方法及其说明如表 7-2 所示。

表 7-2 Handler 的常用方法

方法名称	功能及使用说明
handleMessage(Message msg)	处理消息的方法,在发送消息之后,该方法会自动调用
post(Runnable r)	立即发送 Runnable 对象,该 Runnable 对象最后被封装成 Message 对象
postAtTime(Runnable r, long uptimeMills)	定时发送 Runnable 对象
postDelayed(Runnable r, long delayMills)	延迟发送 Runnable 对象
sendEmptyMessage(int what)	发送空消息
sendMessage(Message msg)	立即发送消息
sendMessageAtTime(Message msg, long uptimeMills)	定时发送消息
sendMessageDelayed(Message msg, long delayMills)	延迟发送消息

重写 Handler 的 handleMessage() 方法,可以实现对消息的处理。如代码段 7-6 所示,可以根据参数选择对此消息是否需要做出处理。

代码段 7-6 重写 Handler 的 handleMessage() 方法,实现对消息的处理

```
Handler mHandler = new Handler() {  
    @Override  
    public void handleMessage(Message msg) {           //重写 handleMessage 方法  
        switch (msg.what) {                           //根据收到的消息的 what 类型处理  
            case BUMP_MSG:  
                Log.v("handler", "Handler 收到消息:" + msg.arg1); //打印收到的消息  
                break;  
            default:  
                super.handleMessage(msg);               //将不需要或者不关心的消息抛给  
                                                         //父类,避免丢失消息  
        }  
    }  
};
```

2. Looper 类

Looper 是用于实现消息队列和消息循环机制的。Looper 负责管理线程的消息队列和消息循环。与 Windows 应用程序的消息处理过程一样,Android 应用程序的消息处理机制也是由消息循环、消息发送和消息处理这 3 个部分组成的,Looper 的作用主要是负责管理消息队列,负责消息的出列和入列操作,执行消息循环。

在消息处理机制中,消息都存放在一个消息队列中,而应用程序的主线程就是围绕这个消息队列进入一个无限循环的,直到应用程序退出。如果队列中有消息,应用程序的主线程就会把它取出来,并分发给相应的 Handler 进行处理;如果队列中没有消息,应用程

以用来保存 int 和 Object 类型的数据,如果要保存其他类型的数据,可以先将要保存的对象放到 Bundle 对象中,然后调用 Message 中的 setData() 方法将 Bundle 对象保存到 Message 对象中。如果一个 Message 只需要携带简单的 int 型信息,应优先使用 Message.arg1 和 Message.arg2 属性来传递信息,这比用 Bundle 对象更节省内存。

7.3.2 使用 Handler 实现线程间通信

使用 Message 机制实现线程间通信主要是为了保证线程之间操作安全,同时不需要关心具体的消息接收者,使消息本身和线程分离,这样就可以方便地实现定时、异步等操作。

实现 Message 机制需要 Handler、Message、Looper 三者之间的互相作用。当线程 A 需要发消息给线程 B 的时候,线程 B 要用自己的 Looper 实例化 Handler 类,即构造 Handler 对象时,把当前线程的 Looper 传给 Handler 构造函数,Handler 对象本身会保存对 Looper 的引用,Handler 对象构造好以后,就可以用其 obtainMessage() 方法实例化 Message 对象,只要把消息数据传递给 Handler,Handler 就会构造 Message 对象,并且把 Message 对象添加到消息队列中。然后就可以调用 Handler 对象的 sendMessage() 方法把 Message 对象发送出去,Looper 就把消息放到消息队列中;最后当 Looper 知道消息队列不为空的时候,就会循环地从消息队列中取消息,若取出消息,就会调用刚才实例化好的 Handler 对象的 handleMessage() 方法处理消息。

如果把 Thread 比作生产车间,那么 Looper 就是放在这个车间里的生产线,这条生产线源源不断地从 MessageQueue 中获取材料 Message,并分发处理 Message。正是因为消息需要在 Looper 中处理,而 Looper 又需运行在 Thread 中,所以不能随随便便地在非 UI 线程中进行 UI 操作。UI 操作通常会通过投递消息来实现,只有往正确的 Looper 投递消息才能得到处理,对于 UI 来说,这个 Looper 一定是运行在 UI 线程中的。

和以是否有无 Looper 来区分 Thread 一样,Handler 的构造函数也分为自带 Looper 和外部 Looper 两大类:如果提供了 Looper,消息会在该 Looper 中处理,否则消息就会在当前线程的 Looper 中处理,当然要确保当前线程一定有 Looper。所有的 UI 线程都是有 Looper 的,所有控件基类的 View 提供了 post 方法,用于向 UI 线程发送消息,并在 UI 线程的 Looper 中处理这些消息。

UI 操作需要向 UI 线程发送消息并在其 Looper 中处理这些消息。这就是为什么不能在非 UI 线程中更新 UI 的原因。控件在非 UI 线程中构造 Handler 时,要么由于非 UI 线程没有 Looper 使得获取 myLooper 失败而抛出 RuntimeException 异常,要么即便提供了 Looper,但这个 Looper 并非 UI 线程的 Looper 而不能处理控件消息。

如上所述,实现这种机制的一般步骤如下:

- (1) 在主线程实例化 Handler,重写 handleMessage() 方法,处理收到的消息。
- (2) 在子线程中实例化 Message 对象。调用已经实例化好的 Handler 对象的 obtainMessage() 方法,把数据传给 obtainMessage() 方法,obtainMessage() 方法就会实例化一个 Message 对象。
- (3) 调用 Handler 的 sendMessage() 方法把已经实例化的 Message 对象发送出去,

添加到 UI 线程的 MessageQueue 中。

(4) UI 线程通过 MainLooper 从消息队列中取出 Handler 发过来的这个消息时,会回调 Handler 的 handleMessage() 方法。

【例 7-3】 示例工程 Demo_07_HandleMessage 演示了线程间的消息机制,如代码段 7-7 所示。

代码段 7-7 线程间的消息机制示例

```
public class MainActivity extends AppCompatActivity {
    private Handler handler;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        handler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                switch (msg.what) {
                    case 1:
                        System.out.println("处理 thread1 的消息:" + msg.arg1);
                        break;
                    case 2:
                        System.out.println("处理 thread2 的消息:" + msg.arg1);
                        break;
                    default:
                        super.handleMessage(msg);
                }
            }
        };
        thread1.start();
        thread2.start();
    }
    Thread thread1 = new Thread(new Runnable() {
        @Override
        public void run() {
            Message msg = handler.obtainMessage();
            msg.what = 1; //线程的标记(int 型)
            msg.arg1 = 6001; //线程的参数
            handler.sendMessage(msg); //发送消息
        }
    });
    Thread thread2 = new Thread(new Runnable() {
        @Override
        public void run() {
```

```

        Message msg = handler.obtainMessage();
        msg.what = 2;           //线程的标记 (int 型)
        msg.arg1 = 6002;       //线程的参数
        handler.sendMessage(msg); //发送消息
    }
};
}

```

从示例程序中可以看出, 一个 Handler 对象可以处理多个发送过来的消息, 通过 Message 中的 what 属性值来区分是哪个线程发送过来的消息。运行结果如图 7-4 所示。



图 7-4 线程间的消息机制

【例 7 4】 示例工程 Demo_07_HandleMessage2 完成与例 7 3 相同的功能, 与之不同的是, 本例采用了另一种实现方法: Handler 对象是在主线程中创建的, 然后通过构造函数传递给线程类。

子线程的定义如代码段 7 8 所示, 重写了包含 Handler 参数的构造方法。创建线程的代码如代码段 7-9 所示。

代码段 7-8 定义子线程类

```

public class Thread1 extends Thread {
    private Handler myhandler;
    public Thread1(Handler handler) {
        myhandler = handler;
    }
    @Override
    public void run() {
        Message msg = myhandler.obtainMessage();
        msg.what = 1;
        msg.arg1 = 1001;
        myhandler.sendMessage(msg);
    }
}

```

代码段 7-9 Handler 对象通过构造函数传递给线程类

```

public class MainActivity extends Activity {
    private Handler handler;
}

```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            if (msg.what == 1) {
                System.out.println("处理 thread1 的消息:" + msg.arg1);
            }
        }
    };
    Thread1 thread = new Thread1(handler);
    thread.start();
}
}

```

【例 7-5】 示例工程 Demo_07_NewThreadDownloadImage 演示了如何使用子线程从网络上异步下载图片并在 ImageView 中显示。

因为需要访问网络,所以要在 AndroidManifest.xml 中添加网络访问权限:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

MainActivity 的布局包括一个下载按钮和一个 ImageView 控件,如代码段 7-10 所示。

代码段 7-10 布局文件

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dip">
    <Button
        android:id="@+id/loadButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="点击下载"/>
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerInside"
        android:padding="2dp"/>
</LinearLayout>

```

Java 源代码如代码段 7-11 所示。其中, mHandle 是主线程也就是 UI 线程处理消息的 Handler 对象;在程序中定义了一个静态方法 loadImageFromUrl(), 实现从网络下载 Bitmap 数据。

代码段 7-11 Java 源代码, 异步下载图片

//package 和 import 语句省略

```
public class MainActivity extends AppCompatActivity {
    private static final String sImageUrl = "http://localhost:8080/myweb/
        flower001.jpg";
    private Button mLoadButton;
    private ProgressDialog mProgressBar;
    private ImageView mImageView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("异步下载图片 - UI thread", ">>onCreate()");
        mProgressBar = new ProgressDialog(this);
        mProgressBar.setMessage("正在下载,请稍候 ...");
        mProgressBar.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
        mProgressBar.setMax(100);
        mImageView = (ImageView)this.findViewById(R.id.imageView);
        mLoadButton = (Button)this.findViewById(R.id.loadButton);
        mLoadButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                mProgressBar.setProgress(0);
                mProgressBar.show();
                new Thread() {
                    @Override
                    public void run() {
                        Log.d("异步下载图片 - Load thread", "开启新线程>>run()");
                        Bitmap bitmap = loadImageFromUrl(sImageUrl);
                        if (bitmap != null) {
                            Message msg = mHandler.obtainMessage(0, bitmap);
                            //mHandler 是主线程(也就是 UI 线程)处理消息的 Handler
                            mHandler.sendMessage(msg);
                        } else {
                            Message msg = mHandler.obtainMessage(1, null);
                            mHandler.sendMessage(msg);
                        }
                    }
                }.start();
            }
        });
    }
}
```

```

    }
    });
}
private Handler mHandler= new Handler() {
//mHandler 是 UI 线程处理消息的 Handler
@Override
public void handleMessage (Message msg) {
    Log.d("异步下载图片 UI thread", ">>handleMessage()");
    switch (msg.what) {
        case 0: //下载成功
            Bitmap bitmap = (Bitmap) msg.obj;
            mImageView.setImageBitmap(bitmap);
            mProgressBar.setProgress(100);
            mProgressBar.setMessage("图片已下载完成!");
            mProgressBar.dismiss();
            break;
        case 1: //下载失败
            mProgressBar.setMessage("图片下载失败!");
            mProgressBar.dismiss();
            break;
    }
}
};
//定义从网络下载 Bitmap 的 static 方法
static Bitmap loadImageFromUrl (String imageUrl) {
    Bitmap bitmap = null;
    try{
        InputStream in = new java.net.URL(imageUrl).openStream();
        bitmap = BitmapFactory.decodeStream(in);
        in.close();
    }catch (Exception e) {
        e.printStackTrace();
    }
    return bitmap;
}
}
}

```

该程序的运行过程是,点击“点击下载”按钮时,会创建一个匿名线程,并调用其 `start()` 方法启动该线程,在这个线程中进行图像下载并解码成 `Bitmap` 格式,然后通过 `Handler` 向 UI 线程发送消息以通知下载结果。UI 线程收到消息之后,会分发给 `Handler`,在它的 `handleMessage()` 方法中根据消息的 ID 来处理下载结果,并相应地更新 UI 界面。

可以从如图 7 5 所示的 LogCat 信息中看到,UI 线程(TID: 7624)和下载线程(TID: 10006)的线程 ID 是不同的,它们是两个独立的线程。


```

        //获取当前内容的总长度
        total = (int)connection.getContentLength();
        //获取输入流
        is = connection.getInputStream();
        bos = new ByteArrayOutputStream();
        byte []data = new byte[1024];
        while((len = is.read(data)) != -1){
            count += len;
            bos.write(data, 0, len);
            //调用 publishProgress() 公布进度
            //onProgressUpdate() 方法将被执行
            publishProgress((int) (count/total * 100));
        }
        bitmap = BitmapFactory.decodeByteArray(bos.toByteArray(), 0,
            bos.toByteArray().length);
        is.close();
        bos.close();
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return bitmap;
}
@Override
protected void onProgressUpdate(Integer... values) {
    super.onProgressUpdate(values);
    mProgressBar.setProgress(values[0]);
}
protected void onPostExecute(Bitmap result) {
    if (result != null) {
        mProgressBar.setProgress(100);
        mProgressBar.setMessage("图片下载完成!");
        mProgressBar.dismiss();
        mImageView.setImageBitmap(result);
    } else {
        mProgressBar.setMessage("图片下载失败!");
        //mProgressBar.dismiss();
    }
}
}
}
}

```

在本例中,首先在任务开始之前在 UI 线程中调用 `onPreExecute()` 方法中设置进度条的初始状态;然后在异步线程中执行 `doInBackground()` 方法以完成下载任务,并在其

中输入的文字。MainActivity 接收这个返回数据,并显示到自己的 TextView 控件中。

示例工程的运行结果如图 8-7 所示。点击 MainActivity 上的“启动 SecondActivity”按钮,则启动第二个 Activity;点击 SecondActivity 界面中的“返回”按钮,则回到 MainActivity,同时在 TextView 控件上显示 SecondActivity 的 EditText 中的文字。

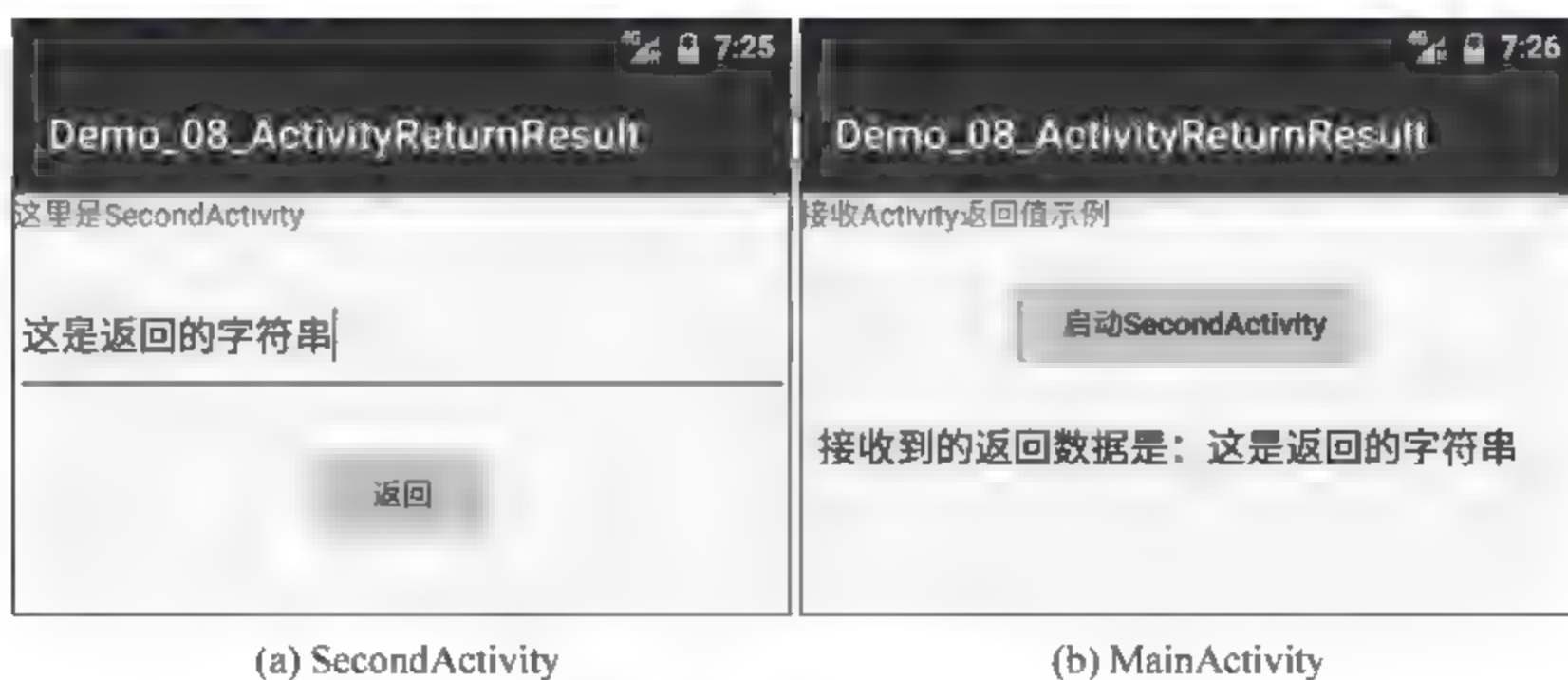


图 8-7 接收 Activity 的返回值

在 SecondActivity 的 XML 布局文件中包括一个 TextView、一个 EditText 和一个“返回”按钮。在 SecondActivity 类中实例化控件,获取 EditText 中输入的文字,处理“返回”按钮的点击事件,主要代码如代码段 8-10 所示。

代码段 8-10 SecondActivity 的主要代码

//package 和 import 语句略

```
public class SecondActivity extends AppCompatActivity {
    private String backstr;
    private EditText txtreturn;
    private TextView input;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        txtreturn = (EditText) findViewById(R.id.txt_return);
        input = (TextView) findViewById(R.id.tv_input);
        Button btnreturn = (Button) findViewById(R.id.btn_return);
        txtreturn.setOnKeyListener(new View.OnKeyListener() {
            @Override
            public boolean onKey(View arg0, int arg1, KeyEvent arg2) {
                backstr=txtreturn.getText().toString();
                //input.setText(backstr);
                return false;
            }
        });
        btnreturn.setOnClickListener(new View.OnClickListener() {
            //按钮对应的点击事件
```

```

        public void onClick(View v) {
            Intent intent = new Intent();
            intent.putExtra("backstring", backstr); //放入返回值
            setResult(0, intent);
                                   //放入回传的值,并添加一个 Code,方便区分返回的数据
            finish();               //结束当前的 Activity,返回
        }
    };
}
}

```

在 MainActivity 的布局文件中包括一个 Button 和两个 TextView,在 MainActivity 类中实例化控件,处理按钮的点击事件。因为要接收 SecondActivity 返回的值,所以跳转的时候调用 startActivityForResult() 方法来启动 SecondActivity,并重写 onActivityResult() 方法接收返回的数据。MainActivity 的主要代码如代码段 8-11 所示。

代码段 8-11 MainActivity 的主要代码

```

//package 和 import 语句略
public class MainActivity extends AppCompatActivity {
    private TextView tvReceive;
    private Intent myintent;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tvReceive = (TextView) findViewById(R.id.tv_return);
        Button btnstart = (Button) findViewById(R.id.btn_1);
        myintent = new Intent();
        myintent.setClass(MainActivity.this, SecondActivity.class);
        btnstart.setOnClickListener(new View.OnClickListener() {
                                   //按钮对应的点击事件
            public void onClick(View v) {
                startActivityForResult(myintent, 1);
                //请求码用于区分请求的数据,只有一个请求时请求码可以为 0
            }
        });
    }
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (data != null) {
            String receive = data.getStringExtra("backstring");
            if (requestCode == 1) { //判断是哪一个控件发出的 Intent 请求
                tvReceive.setText("\n 接收到的返回数据是:" + receive);
            }
        }
    }
}

```


首先创建工程。一般是用一个 Activity 来调用另一个 Service,因此在工程的 src 包中需要编写两个 Java 文件,其中一个是 Service 类,另一个是启动 Service 的主 Activity 类。

创建一个新的 Java 类文件作为 Service。和 Activity 不一样的是,它继承自 `Android.app.Service`。程序开发者如果需要这个 Service 完成什么功能,就在其 `onCreate()`、`onStartCommand()` 中实现。这里在 `onStartCommand()` 中创建并启动了一个新线程,在其中执行相应的操作。主要代码如代码段 8-16 所示。

代码段 8-16 在 Service 中完成的操作

```
//package 和 import 语句略
public class MyService extends Service {
    private boolean running=false;
    @Override
    public void onCreate() {
        super.onCreate();
        running=true;
    }
    //重写 onStartCommand()方法,当此 Service 启动后就会调用该方法
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //创建一个线程并通过 start()运行该线程
        new Thread(){
            @Override
            public void run() {
                super.run();
                while (running){
                    System.out.println("服务已启动,正在运行.....");
                    try {
                        sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }.start();
        return super.onStartCommand(intent, flags, startId);
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        running=false;
        System.out.println("服务已停止.....");
    }
}
```


在 Service 的 onStartCommand()方法中处理接收到的字符串数据,Service 的主要代码如代码段 8-21 所示,运行结果如图 8-12 所示。

代码段 8-21 Service 接收数据

```
//package 和 import 语句略
public class MyService extends Service {
    private boolean running=false;
    String myDataFromActivity;
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        myDataFromActivity= intent.getStringExtra("myData");
        running=true;
        new Thread(){
            @Override
            public void run() {
                super.run();
                while (running){
                    try {
                        System.out.println("服务已启动,接收到数据:"
                            +myDataFromActivity);
                        sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }.start();
        return super.onStartCommand(intent, flags, startId);
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        running= false;
        System.out.println("服务已停止.....");
    }
}
```

8.3.4 将 Service 绑定到 Activity

通过调用 Context.bindService()方法也可以启动 Service,此时 Service 一般依次回调 onCreate()和 onBind()方法完成启动过程。对应的,通过调用 Context.unbindService()方法结束 Service,Service 一般会依次回调 unbind()方法和 onDestroy()方法停止 Service。通过 bindService()方法,Service 就和调用 bindService()的进程“同生共死”了,因此当调


```
Intent serviceIntent = new Intent(MainActivity.this, MyService.class);
bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
```

一旦 Service 被绑定,就可以通过从 onServiceConnected() 处理程序获得的 serviceBinder 对象来使用 Service 所有的公共方法和属性。

【例 8-8】 工程 Demo_08_BindService 演示了 Service 绑定和解除绑定的方法。

首先创建工程,工程的 src 包中包括一个 Service 类和一个 Activity 类。在 Service 的 onBind() 中创建并启动了一个新线程,在其中执行相应的操作。主要代码如代码段 8-23 所示。

代码段 8-23 在 Service 中完成的操作

```
//package 和 import 语句略
public class MyService extends Service {
    private boolean running = false;
    public MyService() {
    }
    @Override
    public void onCreate() { //这个方法在 Service 创建时被调用
        super.onCreate();
        Log.d("我的提示", "MyService:onCreate()被调用");
        running = true;
    }
    @Override
    public IBinder onBind(Intent intent) { //成功绑定后调用该方法
        Log.d("我的提示", "MyService:onBind()被调用");
        running = true;
        new Thread() {
            @Override
            public void run() {
                super.run();
                while (running) {
                    try {
                        System.out.println("服务已启动.....");
                        sleep(2000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }.start();
        return new Binder();
    }
    @Override
```



```
    }  
};  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    findViewById(R.id.btnBindService).setOnClickListener(new View.  
        OnClickListener() {  
        @Override  
        public void onClick(View view) {  
            Intent serviceIntent = new Intent(MainActivity.this, MyService.  
                class);  
            bindService(serviceIntent, serviceConnection, Context.BIND_  
                AUTO_CREATE);  
            //绑定 Service  
        }  
    });  
    findViewById(R.id.btnUnbindService).setOnClickListener(new View.  
        OnClickListener() {  
        @Override  
        public void onClick(View view) {  
            unbindService(serviceConnection);           //解除绑定 Service  
        }  
    });  
}  
}
```

程序的运行结果如图 8-14 所示,从输出结果也可以看到 Service 被绑定和解除绑定时回调方法的调用过程。

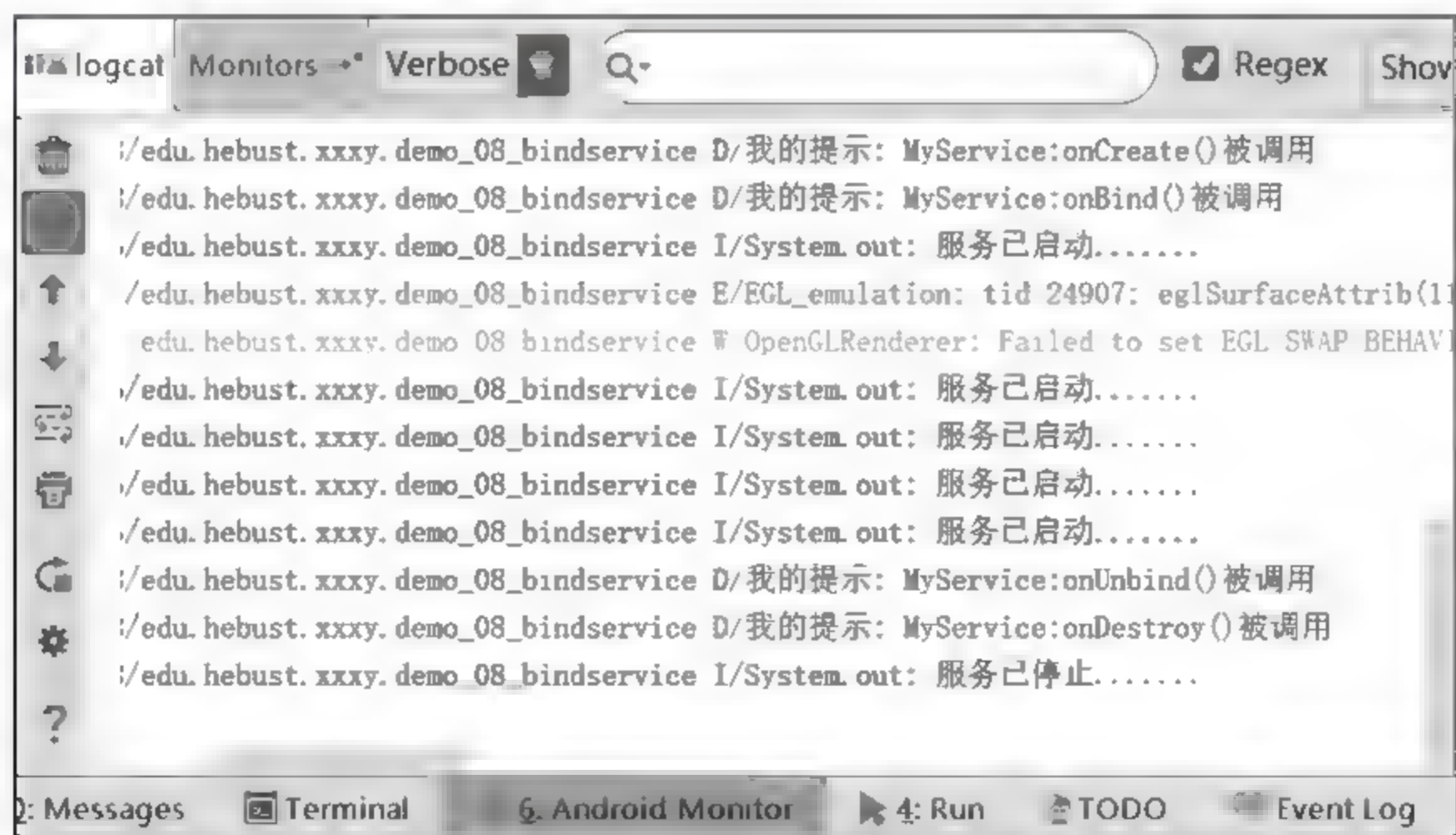


图 8-14 绑定和解除绑定 Service

代码段 8-25 在 AndroidManifest.xml 中完成静态注册

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="BroadcastReceiverDemo"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity
            android:name=".BroadcastReceiverDemo"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <receiver android:name=".MyBroadcastReceiver">
            <intent-filter>
                <action android:name="BroadcastReceiverDemo"/>
            </intent-filter>
        </receiver>
    </application>
</manifest>

```

【例 8 9】 工程 Demo_08_BroadcastReceiverXML 演示了广播消息的发送和接收。本例使用静态注册的广播接收器接收广播消息,并在 LogCat 中显示接收到的消息。

首先,新建工程,然后在包中创建继承自 BroadcastReceiver 类的 MyBroadcastReceiver 类并重写 onReceive() 方法,如代码段 8-26 所示。

代码段 8-26 定义 BroadcastReceiver

```

//package 和 import 语句略
public class MyBroadcastReceiver extends BroadcastReceiver {
    private static final String TAG = "MyBroadcastReceiver";
    public MyBroadcastReceiver() {
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(TAG, "接收器接收了广播");
        String receive action = intent.getAction();           //获取广播的 Action
        Log.d(TAG, "接收器收到广播的 action:" + receive action);
        String receive message = intent.getStringExtra("message");
        Log.d(TAG, "接收器收到广播的 message:" + receive message);
    }
}

```


为简便起见,这里采用静态注册的方法,因此需要在工程的 AndroidManifest.xml 中完成对相应的多个基于 BroadcastReceiver 的类的说明。由于这里要向多个 BroadcastReceiver 同时发送广播,因此在说明这些 BroadcastReceiver 对象时,要保证它们拥有相同的<intent-filter>,即保证<action>元素的配置信息相同。为了接收有序广播,这里设置了接收器的优先级分别为 1 和 2,NewBroadcastReceiver 拥有更高的优先级,如代码段 8-32 所示。

代码段 8-32 在 AndroidManifest.xml 中完成静态注册

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.hebust.xxy.demo_08_sendorderedbroadcast">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <receiver
            android:name=".MyBroadcastReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                android:priority="1">
                <action android:name="hebust.xxy.intent.action.MYBROADCAST"/>
            </intent-filter>
        </receiver>
        <receiver
            android:name=".NewBroadcastReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                android:priority="2">
                <action android:name="hebust.xxy.intent.action.MYBROADCAST"/>
            </intent-filter>
        </receiver>
    </application>
</manifest>
```


8.5 本章小结

本章介绍了 Intent、Service 和 BroadcastReceiver 的概念及其应用。Intent 是 Android 系统的消息传递机制,用于实现 Activity、Service、BroadcastReceiver 等组件之间的交互和通信。Service 是运行在后台的长生命周期的、没有 UI 界面的 Android 组件。BroadcastReceiver 是对系统中的广播消息进行过滤、接收并响应的一类组件。要学好本章内容,需要熟练掌握 Intent 的概念和用法,因为在启动其他 Activity、启动 Service、发送广播消息、传递数据时都需要用到 Intent。

习 题

1. 什么是 Service? Service 与 Broadcast 有什么不同?
2. 调用 startService() 和 bindService() 启动服务有什么区别?
3. 在一个 Service 对象的生命周期内,Service 对象会多次调用 onCreate() 方法吗? 会多次调用 onStartCommand() 方法吗?
4. 利用 Service 实现一个音乐播放器 MusicBox,要求如下:
 - (1) 采用 XML 文件实现布局,Activity 中有一个 TextView 用于显示正在播放的歌曲名称,一个 ListView 用于显示播放文件列表,还有一个 start 按钮和一个 stop 按钮。
 - (2) 点击 start 按钮运行服务(播放音乐),点击 stop 按钮停止服务(停止播放音乐),并将歌曲名称显示在 Activity 中。
5. 在第 4 题中添加一个音乐播放进度条,并实现拖动播放功能。
6. 什么是 Broadcast? 描述它的 3 种发送方式的不同之处。
7. 设计一个应用程序,要求用户输入用户名和密码。当用户输入正确的用户名和密码,点击“登录”按钮后,发送广播消息“有用户登录入系统!”;当用户输入用户名和密码有误,点击“登录”按钮后,发送广播消息“有非法用户试图登录入系统,被拒绝!”
8. 设计一个 BroadcastReceiver 并启动它,接收第 7 题中的广播消息。

相关代码如代码段 9-1 所示。

代码段 9-1 读写 SharedPreferences 信息

```
public class MainActivity extends AppCompatActivity {  
    EditText myUsername, myPassword;           //输入的用户名、密码  
    static final String KEY1 = "userName";  
    static final String KEY2 = "userPass";     //存入 SharedPreferences 中的 Key  
    SharedPreferences preferences;            //定义 SharedPreferences 对象  
    TextView tvRead;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        myUsername = (EditText)findViewById(R.id.etusername);  
        myPassword = (EditText)findViewById(R.id.etpassword);  
        tvRead = (TextView)findViewById(R.id.tvRead);  
        preferences = getPreferences(Activity.MODE_PRIVATE);  
                                                //获取 SharedPreferences 对象  
        final SharedPreferences.Editor editor = preferences.edit();  
        findViewById(R.id.btnSave).setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                //将用户输入的 EditText 信息存储到 SharedPreferences 中  
                editor.putString(KEY1, myUsername.getText().toString());  
                //两个参数分别是键和值  
                editor.putString(KEY2, myPassword.getText().toString());  
                editor.commit();  
            }  
        });  
        findViewById(R.id.btnread).setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {           //由于不编辑,这里不用 Editor 对象  
                String name = preferences.getString(KEY1, "当前数据不存在");  
                //获取指定键的值,第二个参数是当第一个参数不存在时,为其指定默认值  
                String pass = preferences.getString(KEY2, "当前数据不存在");  
                tvRead.setText("从 SharedPreferences 读出的数据:\n"  
                    + "\n 用户名:" + name + "\n 密码:" + pass);  
            }  
        });  
        findViewById(R.id.btnquit).setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                finish();  
            }  
        });  
    }  
}
```


过程的 `openFileInput()` 方法和 `openFileOutput()` 方法,前者为读取数据做准备而打开应用程序私有文件,后者为写入数据做准备而打开应用程序私有文件。所以,在 Android 系统中,读写内部文件不用自己去创建文件对象和输入输出流,提供文件名就可以返回 File 对象或输入输出流。

1. 从文件中读取数据

如果要打开应用程序的私有文件并读取其中的数据,可以使用标准数据输入流。通过调用 Activity 的 `openFileInput()` 方法可以获得标准数据输入流对象,方法的定义如下:

```
public FileInputStream openFileInput (String name)
```

该方法的返回值是一个 `FileInputStream` 对象,这是字节流,对于文本文件的读出并不方便,所以通常使用 `InputStreamReader` 将其进一步包装成为字符流,再调用其 `read()` 方法将字符串读出。代码段 9-3 是一个读取文件的示例,`openFileInput()` 方法中的参数是准备读出数据的文件名,这里文件名不能包含路径分隔符“/”。操作完成后要调用 `close()` 方法关闭输入流。

代码段 9-3 从文件中读取数据

```
public class FileActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        try {  
            //获取文件输入流  
            FileInputStream inStream = this.getContext().openFileInput  
                ("fileName.txt");  
            //包装为字符流  
            InputStreamReader inStreamReader = new InputStreamReader  
                (inStream, "UTF-8");  
            //用输入流的实际长度来构建字符数组,读取到字符数组  
            char myContent[] = new char[inStream.available()];  
            inStream.read(myContent);  
            //将前述得到的字符数组转换到字符串中  
            String listResult = new String(myContent);  
            inStreamReader.close();  
            inStream.close();  
        } catch (Exception e) {  
            //异常处理  
        }  
    }  
}
```



```

//在其中输入的信息将存到内部文件中
etFileName = (EditText) findViewById(R.id.etFileName);
//内部文件的文件名
tvRead = (TextView) findViewById(R.id.textRead);
//显示从内部文件中读取的信息
findViewById(R.id.btnSave).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { //将 EditText 中输入的信息写入内部文件中
        myfilename=etFileName.getText().toString()+ ".txt";
        try {
            FileOutputStream fileOutputStream = openFileOutput(myfilename,
                Context.MODE_PRIVATE);
            OutputStreamWriter outputStreamWriter = new OutputStreamWriter
                (fileOutputStream, "UTF-8");
            outputStreamWriter.write(etWriteText.getText().toString());
            //按照指定编码方式写入 OutputStreamWriter
            outputStreamWriter.flush();
            fileOutputStream.flush();
            outputStreamWriter.close();
            fileOutputStream.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        Toast.makeText(getApplicationContext(), "写入完成", Toast.
            LENGTH_LONG).show();
    }
});
findViewById(R.id.btnRead).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { //读取数据
        myfilename=etFileName.getText().toString()+ ".txt";
        try {
            FileInputStream fileInputStream = openFileInput(myfilename);
            //得到的是字节流,然后包装为字符流
            InputStreamReader inputStreamReader = new InputStreamReader
                (fileInputStream, "UTF-8");
            char mycontent[] = new char[fileInputStream.available()];
            inputStreamReader.read(mycontent);
            inputStreamReader.close();
            fileInputStream.close();
            String listResult = new String(mycontent);
        }
    }
});
```


用不存在了,这些照片也不会被删除。而对于私有类型的文件,由于是外部存储的原因,它们也能被其他程序访问,只不过一个私有文件对其他应用来说通常没有访问价值。对于应用程序来讲,在外部存储上使用私有文件的好处是,当应用程序被卸载之后,这些文件也会被删除。

如果想在外部存储上存储公共文件,可以调用 `getExternalStoragePublicDirectory()` 方法获取存储路径。代码段 9-7 获得了存放 picture 的目录,并且创建了一个新文件。代码中 `Environment.DIRECTORY_PICTURES` 的值就是字符串 `picture`。

代码段 9-7 读写内部文件

```
public File getAlbumStorageDir(String albumName) {
    File file = new File(Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_PICTURES), "new_image.jpg");
    if(!file.mkdirs()) {
        Log.e(LOG_TAG, "Directory not created");
    }
    return file;
}
```

需要注意的是,对于不同设备和 Android 版本,应用程序的外部存储路径会有所不同,获取外部存储路径的方法也不相同。如果 Android 版本低于 API level 8,那么不能通过调用 `Environment.getExternalStoragePublicDirectory()` 方法获取存储路径,而是通过调用 `Environment.getExternalStorageDirectory()` 方法获取,该方法不带参数,即不能自己创建一个目录,只是返回外部存储的根路径。

在使用外部存储之前,必须先调用 `Environment.getExternalStorageState()` 方法来检查外部存储设备的当前状态,以判断其是否可用。代码段 9-8 是一个示例,这个例子只检查了外部存储设备是否可读写,它还有很多其他的状态,例如与计算机连接、没有设备等,可根据程序需求用类似的方法检测。

代码段 9-8 检查外部存储的当前状态

```
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)) { //外部存储可以读写
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} elseif (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    //外部存储是只读的
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else { //其他错误状态
    mExternalStorageAvailable = mExternalStorageWriteable = false;
}
```


该应用程序的外部存储权限许可,如图 9-8 所示。

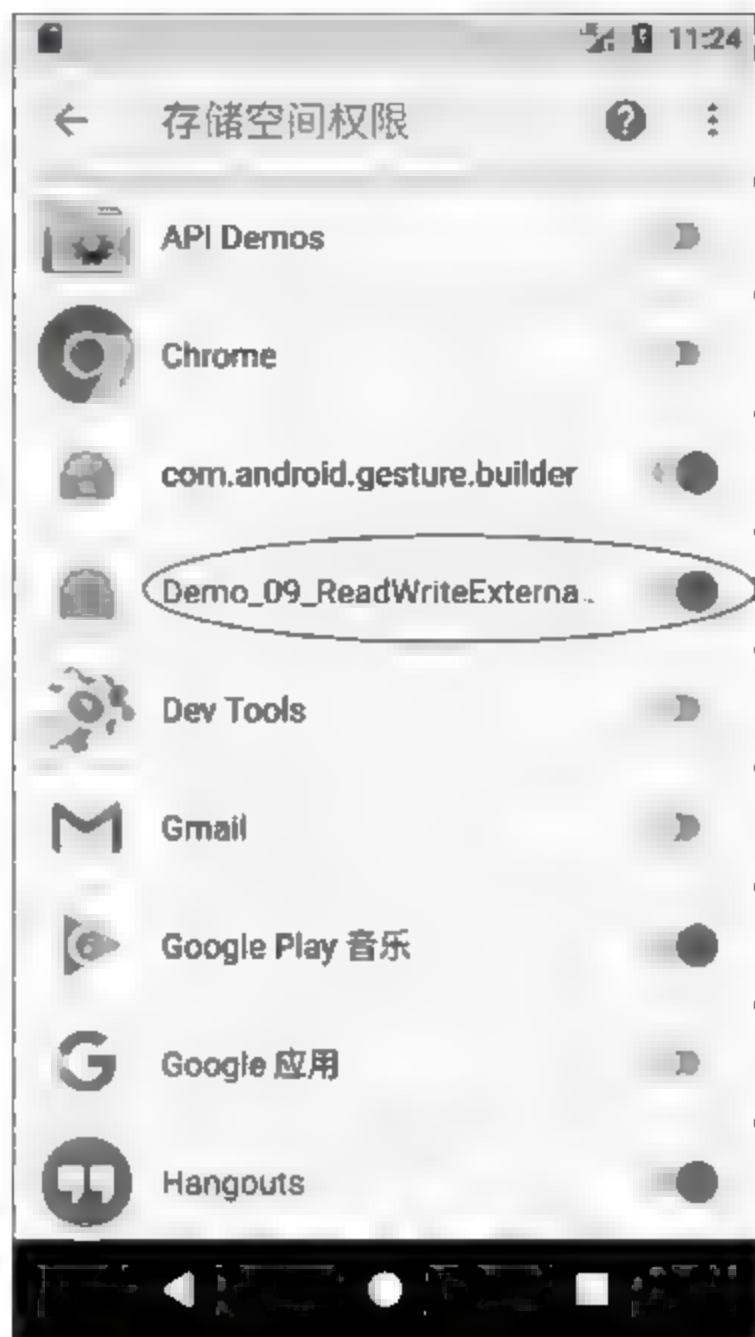


图 9-8 在手机中设置应用程序的外部存储读写许可

程序的运行结果如图 9-9 所示。

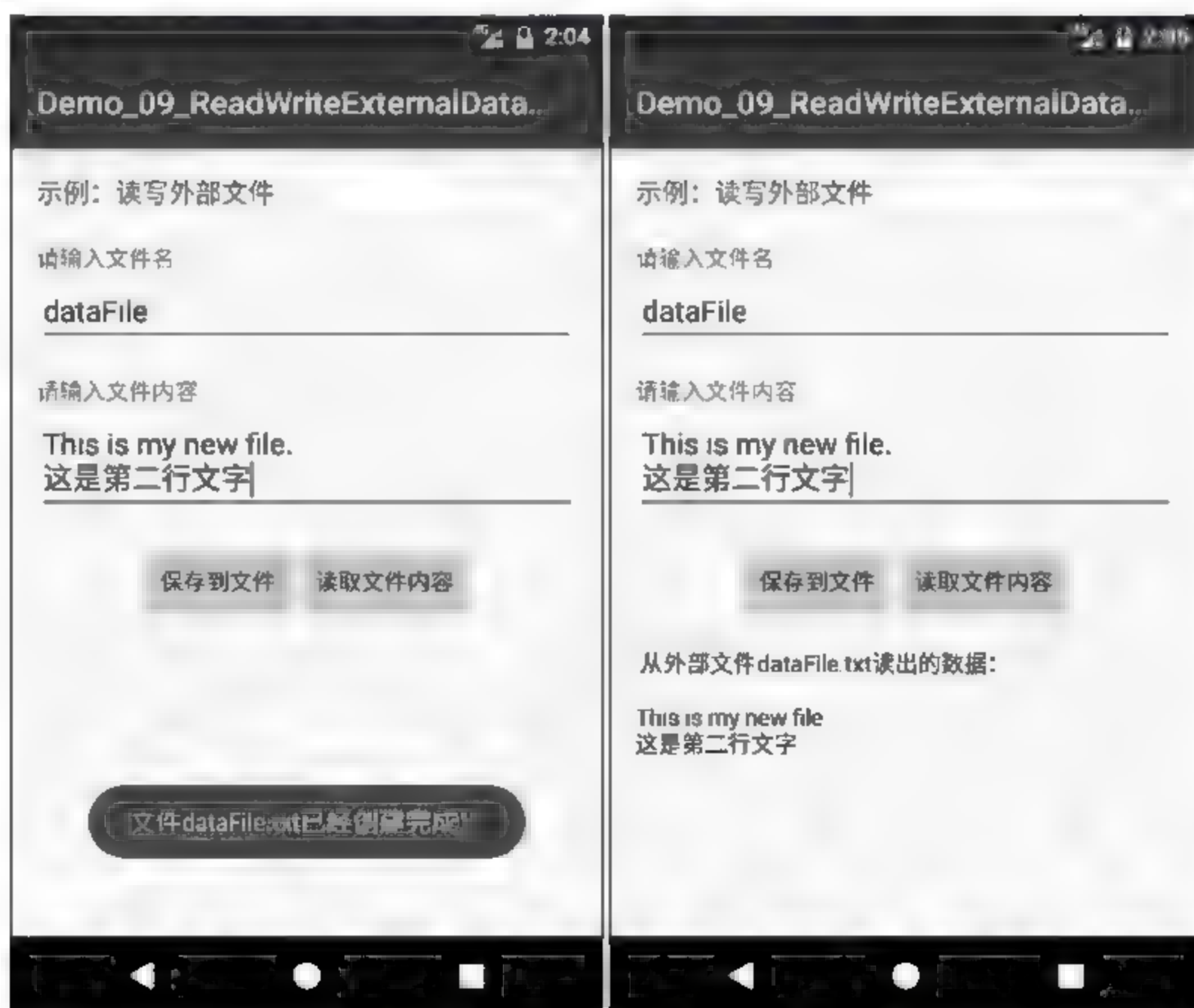


图 9-9 外部数据文件读写的示例

程序代码如代码段 9-10 所示,保存数据时首先获得外部存储的工作路径,然后创建文件对象,判断外部存储是否可用,可用则创建文件,写入数据。

代码段 9-10 读写外部存储设备中的文件

```
//package 和 import 语句略
public class MainActivity extends AppCompatActivity {
    EditText etWriteText,etFileName;
    TextView tvRead;
    private File sdCard = Environment.getExternalStorageDirectory();
                                                                    //获取外存路径

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        etWriteText = (EditText)findViewById(R.id.etFileText);
                                                                    //在其中输入的信息将存到外部文件中
        etFileName = (EditText)findViewById(R.id.etFileName);
                                                                    //外部文件的文件名
        tvRead = (TextView)findViewById(R.id.textRead);
                                                                    //显示从外部文件中读取的信息
        findViewById(R.id.btnSave).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { //将 EditText 中输入的信息写入外部文件中
                tvRead.setText("");
                String myfilename=etFileName.getText().toString()+".txt";
                try {
                    File newFile =new File(sdCard, myfilename);
                                                                    //存储到外存根目录

                    if (sdCard.exists()) {
                        newFile.createNewFile();
                        Toast.makeText(getApplicationContext(), "文件"+myfilename
                            + "已经创建完成!", Toast.LENGTH_LONG).show();
                        FileOutputStream fos =new FileOutputStream(newFile);
                                                                    //打开文件输出流(字节流)
                        OutputStreamWriter osw =new OutputStreamWriter(fos,
                            "UTF 8");
                                                                    //包装成字符流
                        osw.write(etWriteText.getText().toString());
                        osw.flush();
                        fos.flush();
                        osw.close();
                        fos.close();
                        Toast.makeText(getApplicationContext(), "数据已经成功写入
```

```
        外部文件!" + myfilename, Toast.LENGTH_SHORT).show();
    }else{
        Toast.makeText(getApplicationContext(), "不存在外部存
            储路径!", Toast.LENGTH_LONG).show();
        return;
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
});

findViewById(R.id.btnRead).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { //读取数据
        String myfilename = etFileName.getText().toString() + ".txt";
        try {
            File newFile = new File(sdCard, myfilename);
            if (newFile.exists()) {
                FileInputStream fis = new FileInputStream(newFile);
                //读取文件中的数据,得到的是字节流
                InputStreamReader isr = new InputStreamReader(fis, "UTF-8");
                //包装为字符流
                char mycontent[] = new char[fis.available()];
                //用 fis 的实际长度来构建字符数组
                isr.read(mycontent); //读取
                isr.close();
                fis.close();
                String listResult = new String(mycontent);
                //将前述得到的字符数组转换到字符串中
                tvRead.setText("从外部文件" + myfilename + "读出的数据:\n\n"
                    + listResult);
                //将读取到的内容展现在 TextView 上
            }else{
                Toast.makeText(getApplicationContext(), "文件不存在!",
                    Toast.LENGTH_LONG).show();
                return;
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
```


SQLiteDatabase 的 Close()方法关闭数据库。

9.3.3 创建数据库和数据表

在 Android 中,SQLite 数据库文件存储在 data/data/<当前包名>/databases/ 下。默认状态下,该数据库文件只能由创建它的应用程序使用。

SQLite 和其他数据库最大的不同就是对数据类型的支持,创建一个数据表时,可以在 CREATE TABLE 语句中指定某列的数据类型,也可以把任何数据类型放入任何列中。当某个值插入数据库时,SQLite 将检查它的类型。如果该类型与关联的列不匹配,则 SQLite 会尝试将插入值转换成该列的类型。如果不能转换,则插入值将作为其本身具有的类型存储。例如,可以把一个字符串(String)放入 INTEGER 数据类型的列。这种特性称为“弱类型”。

SQLite 将数据值的存储划分为 5 种类型,如表 9-3 所示。

表 9-3 SQLite 的数据类型

数据类型	说 明
NULL	表示该值为 NULL 值
INTEGER	带符号整型值
REAL	浮点值
TEXT	文本字符串,存储使用的编码方式为 UTF-8、UTF-16BE、UTF-16LE
BLOB	二进制对象,该类型数据和输入数据完全相同

由于 SQLite 采用的是动态数据类型,而其他传统的关系型数据库使用的是静态数据类型,即字段可以存储的数据类型是在创建数据表时必须确定的,因此它们之间在数据存储方面还是存在较大的差异。在 SQLite 中,存储分类和数据类型也有一定的差别,如 INTEGER 存储类别可以包含 6 种不同长度的整型数据类型,然而这些 INTEGER 数据一旦被读入到内存后,SQLite 会将其全部视为占用 8B 的整型。因此对于 SQLite 而言,即使在数据表中定义了明确的字段类型,仍然可以在该字段中存储其他类型的数据。然而需要特别说明的是,尽管 SQLite 为我们提供了这种方便,但是考虑到数据库平台的可移植性问题,在实际的开发中还是应该尽可能保证数据类型的存储和声明的一致性。

另外,SQLite 没有提供专门的布尔存储类型,取而代之的是整型 1 表示 true,0 表示 false。

SQLite 也同样没有提供专门的日期时间存储类型,而是以 TEXT、REAL 和 INTEGER 类型分别以不同的格式表示日期时间。TEXT 类型采用“YYYY MM DD HH:MM:SS.SSS”格式存储日期时间;REAL 类型以 Julian 日期格式存储,即自格林威治时间公元前 4713 年 1 月 1 日中午以来的天数;INTEGER 类型以 UNIX 时间形式保存数据值,即从 1970-01-01 00:00:00 到当前时间的毫秒数。

【例 9-6】 示例工程 Demo_09_CreateDatabase 演示了如何创建数据库并在新建的数据库中创建数据表。

首先,新建一个类 MyDBOpenHelper,该类必须继承自 SQLiteOpenHelper,如代码段 9-11 所示。

代码段 9-11 创建 SQLiteOpenHelper 的子类

```
//package 和 import 语句略
public class MyDBOpenHelper extends SQLiteOpenHelper {
    public MyDBOpenHelper(Context context, String name, SQLiteDatabase.
        CursorFactory factory, int version) {
        //重写构造方法,创建数据库文件
        super(context,name,factory, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        //执行 SQL 语句,创建数据表
        db.execSQL("CREATE TABLE student(" +
            "_id INTEGER PRIMARY KEY AUTOINCREMENT," +
            "stuId INTEGER UNIQUE," +
            "stuName TEXT NOT NULL," +
            "stuClass TEXT);");
        //初始化一些数据
        db.execSQL("INSERT INTO student (stuId, stuName, stuClass) values
            (31,'李国庆','软件 151')");
        db.execSQL("INSERT INTO student (stuId, stuName, stuClass) values
            (32,'刘凯旋','软件 151')");
    }
    @Override
    public void onUpgrade(SQLiteDatabase _db, int oldVersion, int newVersion) {
        //数据库需要升级时被调用
        _db.execSQL("DROP TABLE IF EXISTS student");
        onCreate(_db);
    }
}
```

通常需要重写 MyDBOpenHelper 的 3 个方法:构造方法、onCreate()方法、onUpgrade()方法。

SQLiteOpenHelper 类要求必须重写其构造方法。构造方法有多种重载形式,重写其中一个即可。通常重写时会调用父类的构造方法 SQLiteOpenHelper(Context context,String name,CursorFactory factory,int version)创建一个数据库文件,该构造方法需要 4 个参数:上下文环境(如 Activity)、数据库名字、游标工厂(通常是 null)、代表正在使用的数据库模型版本的整数。

回调 onCreate()方法时会传入一个 SQLiteDatabase 对象,可以根据需要在这个数据库中创建数据表和初始化数据。数据库第一次创建的时候会调用这个方法,可以通过调用 SQLiteDatabase 对象的 execSQL()方法来执行 SQL 语句,完成创建表和索引的过程,

如果没有异常,这个方法没有返回值。

onUpgrade()方法定义了3个参数,分别是 SQLiteDatabase 对象、旧的版本号、新的版本号。当数据库需要升级的时候,即调用构造方法时传入的版本号发生了变化,Android 系统会主动地调用 onUpgrade()方法。一般在这个方法中删除旧数据库表,并建立新的数据库表。当然,是否还需要做其他的操作,完全取决于应用程序的需求。

创建完成 SQLiteOpenHelper 的子类后,在 Activity 中实例化这个类,就可以创建相应的数据库了,代码段 9-12 是一个示例。

代码段 9-12 实例化 MySQLiteOpenHelper 类,创建并初始化数据库

```
//package 和 import 语句略
public class MainActivity extends AppCompatActivity {
    private MyDBOpenHelper dbOpenHelper;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //实例化 SQLiteOpenHelper 的子类,传入数据库名称(SC_Database.db)、版本号
        //创建相应的数据库和数据表
        dbOpenHelper=new MyDBOpenHelper(getApplicationContext(),
            "SC_Database.db", null, 1);
    }
}
```

当实例化 SQLiteOpenHelper 的子类时,会调用其构造方法创建数据库,代码段 9-12 创建了名为 SC_Database.db 的数据库文件。如果是第一次创建数据库,会调用 onCreate()方法,创建表和索引。

与文件存取方式类似,SQLite 数据库文件存储在/data/data/<当前包名>/databases 目录中,如图 9-10 所示。默认状态下,该数据库文件只能由创建它的应用程序使用。其他 Activity 可以通过 ContentProvider 访问这个数据库。

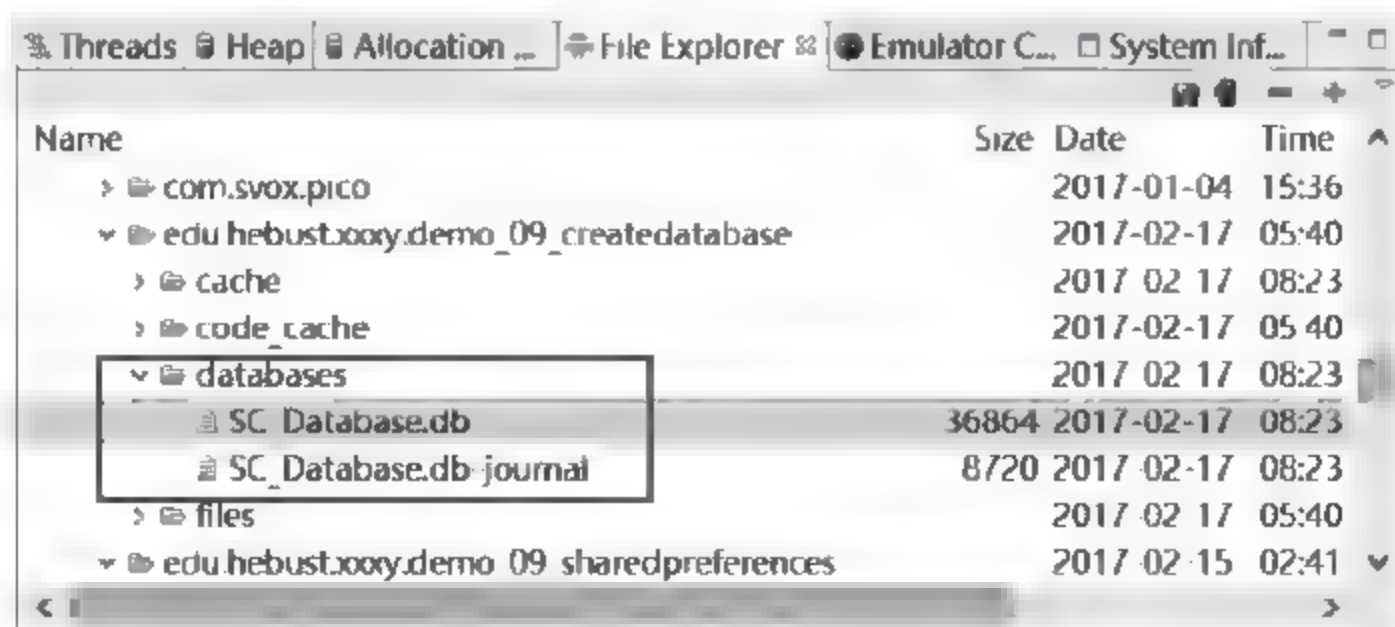


图 9-10 创建的数据库文件

9.3.4 操纵数据库中的数据

创建了数据库之后,可以使用 SQLiteOpenHelper 类的 getReadableDatabase() 或 getWritableDatabase() 方法得到 SQLiteDatabase 实例对象。获得了 SQLiteDatabase 对象以后,就可以通过调用 SQLiteDatabase 的实例方法来对数据库进行增、删、改、查。对数据库的操作结束后,需要调用 SQLiteDatabase 的 Close() 方法来关闭数据库。

1. 查询数据

调用 getReadableDatabase() 方法可以得到对数据库具有读的权限的 SQLiteDatabase 实例对象,调用该对象的 query() 方法,可以对数据库中的数据进行查询操作。该方法返回一个 Cursor 对象,定义如下:

```
Cursor query(String table, String[] columns, String selection, String[]  
selectionArgs, String groupBy, String having, String orderBy, String limit);
```

query() 方法将 SELECT 语句的内容定义为各参数,除了数据表名,其他参数都可以是 null。方法中的参数意义如下:

- (1) table: 查询数据表的表名,不可为 null。
- (2) columns: 按查询要求返回的列名数组,如果其值为 null 则返回所有列。
- (3) selection: 查询的条件表达式,相当于 SQL 语句的 WHERE 子句,格式形如 “_id=?”,其中的问号是占位符,供下一个参数 selectionArgs 填充,如果其值为 null,则返回所有的行。
- (4) selectionArgs: 查询条件所需的参数值,该数组的值依次填充 selection 参数中的每一个问号。
- (5) groupBy: 定义查询是否分组,相当于 SQL 语句中的 GROUP BY 子句,如果其值为 null,则不分组。
- (6) having: 分组条件表达式,相当于 SQL 语句中的 HAVING 短语,和 GROUP BY 子句配套使用,表示对分组的筛选条件,如果 having 值为 null,则保留所有的分组。
- (7) orderBy: 查询结果排序依据的列,相当于 SQL 语句中的 ORDER BY 子句,描述对查询结果的排序要求,如果 orderBy 值为 null,将会使用默认的排序规则。
- (8) limit: 限定查询返回的行数。如果其值为 null,将返回所有行。

【例 9.7】 示例工程 Demo_09_QueryDatabase 演示了查询数据库并将查询结果的数据显示在 ListView 中。

示例中调用 SQLiteDatabase 对象的 query() 方法实现数据库的查询,返回所有班级为“软件 151”的数据,运行结果如图 9-11 所示。

示例程序中使用 SimpleCursorAdapter 为 ListView 对象提供数据源,相关代码如代码段 9-13 所示。



The screenshot shows an Android application window with the title "Demo_09_QueryDatabase". Inside the window, there is a text label "示例：查询数据表" (Example: Query Data Table). Below the label is a table with three columns: "学号" (Student ID), "姓名" (Name), and "班级" (Class). The table contains six rows of data.

学号	姓名	班级
31	李国庆	软件151
33	赵坤	软件151
35	赵丽芳	软件151
36	马红艳	软件151
38	马克	软件151
39	赵丽娟	软件151

图 9-11 显示查询的结果

代码段 9-13 查询数据

//package 和 import 语句略

```

public class MainActivity extends AppCompatActivity {
    private SQLiteDatabase dbRead;
    private MyDBOpenHelper dbOpenHelper;
    private SimpleCursorAdapter listViewAdapter;
    private ListView listView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listView= (ListView) findViewById(R.id.listView);
        //实例化 SQLiteOpenHelper 的子类,创建数据库和数据表
        dbOpenHelper=new MyDBOpenHelper (getApplicationContext(),
            "SC_Database.db", null, 1);
        dbRead =dbOpenHelper.getReadableDatabase();
        Cursor result =null;
        result =dbRead.query("student", null,"stuClass=?", new String[]
            {"软件 151"}, null, null, null);
        listViewAdapter=new SimpleCursorAdapter (getApplicationContext(),R.
            layout.item_list,
            result,new String[]{"stuId", "stuName", "stuClass"},
            new int[]{R.id.itemID,R.id.itemName,R.id.itemClass},
            CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);
        listView.setAdapter(listViewAdapter);
    }
}

```

query()方法返回的是一个 Cursor 对象,其游标最开始指向的是查询结果集合中第一行的上一行。如果需要在程序中使用某一条数据,则应该首先调用 Cursor 对象 next()

方法将游标移动到记录集合的第一行,接着再获取数据即可。代码段 9-14 遍历了 student 表。

代码段 9-14 遍历 student 表

```
DBOpenHelper dbOpenHelper = new DBOpenHelper(getApplicationContext(),
    "SC_Database.db", null, 1);
SQLiteDatabase dbRead = dbOpenHelper.getReadableDatabase();
Cursor result = dbRead.query("student", null, null, null, null, null, null);
result.moveToFirst();
while (!result.isAfterLast()) {
    int stuID = result.getInt(1);
    String stuName = result.getString(2);
    String stuClass = result.getString(3);
    result.moveToNext();
}
result.close();
```

实现数据库的查询,也可以通过调用 SQLiteDatabase 对象的 rawQuery() 方法实现。该方法执行一条由字符串描述的 SELECT 语句,返回值是也一个 Cursor 对象,如代码段 9-15 所示。

代码段 9-15 调用 rawQuery() 方法对数据库查询

```
DBOpenHelper dbOpenHelper = new DBOpenHelper(getApplicationContext(),
    "SC_Database.db", null, 1);
SQLiteDatabase dbRead = dbOpenHelper.getReadableDatabase();
Cursor result = dbRead.rawQuery("SELECT * FROM student WHERE stuClass=
    '软件 151'", null);
```

2. 插入数据

调用 getWritableDatabase() 方法可以获得具有写入权限的 SQLiteDatabase 对象,再调用 SQLiteDatabase 对象的 insert() 方法,实现数据的插入。

insert() 方法的定义如下:

```
long insert(String table, String nullColumnHack, ContentValues values)
```

与 query() 方法类似,insert() 方法把 SQL 语句的各部分作为参数值传入。各参数含义如下:

- (1) table: 想要插入数据的表名,不可为 null。
- (2) nullColumnHack: 由于 SQL 标准不允许插入空行,当初始化值为空时,这一列将会被显式地赋一个 null 值。
- (3) values: 要插入的值。

当插入数据时,如果 values 参数值为 null 或者元素个数为 0,因为数据库不允许插入

一个空行,插入操作会失败。为了防止出现这种情况,在 insert()方法的第二个参数指定一个列名,如果将要插入的行为空行时,系统会将指定的这个列的值设为 null,然后再向数据库中插入。

向数据库的表中插入记录时,需要先将数据包含在一个 ContentValues 对象中。ContentValues 是一个数据承载容器,使用方法是先创建一个 ContentValues 对象,然后调用其 put()方法向该对象插入键-值对,其中键名必须是数据表中的列名,值是希望插入到这一列的值,而且值的类型要和数据库中的数据类型一致。

insert()方法的返回值是新添记录的行号,与主键 id 的值无关。代码段 9-16 是实现插入数据的一个示例。

代码段 9-16 插入数据

```
ContentValues cv=new ContentValues();  
cv.put("stuId", 32); //将值存放到对应的键中  
cv.put("stuName", "刘凯旋"); //将值存放到对应的键中  
cv.put("stuClass", "软件 152"); //将值存放到对应的键中  
dbWrite.insert("student ", "stuId", cv); //执行插入,返回新添记录的行号
```

3. 删除数据

调用 SQLiteDatabase 对象的 delete()方法可以删除数据表中的数据。delete()方法的定义如下:

```
int delete(String table, String whereClause, String[] whereArgs)
```

该方法的各参数含义如下:

- (1) table: 想要删除数据的表名,不可为 null。
- (2) whereClause: 是可选的 WHERE 子句,格式形如“_id=?”,其中的问号是占位符,供下一个参数 whereArgs 填充。如果其值为 null,将会删除所有的行。
- (3) whereArgs: 删除条件所需的参数值,该数组中的值依次填充 whereClause 参数中的每一个问号。

代码段 9 17 是删除数据的一个示例,本例中删除了 student 表中的全部数据。

代码段 9-17 删除数据

```
DBOpenHelper dbOpenHelper =new DBOpenHelper(getApplicationContext(),  
    "student.db", null, 1);  
SQLiteDatabase dbWriter=dbOpenHelper.getWritableDatabase();  
dbWriter.delete("student",null,null);  
dbWriter.close();
```

4. 修改数据

调用 SQLiteDatabase 对象的 update()方法可以修改数据表中的数据。update()方

法会根据条件修改指定列的值,定义如下:

```
int update(String table, ContentValues values, String whereClause, String[]  
    whereArgs)
```

该方法的各参数含义如下:

- (1) table: 想要修改数据的表名,不可为 null。
- (2) values: 要更新的值。
- (3) whereClause: 是可选的 WHERE 子句,格式形如“_id=?”,其中的问号是占位符,供下一个参数 whereArgs 填充。如果其值为 null,将会修改所有的行。
- (4) whereArgs: 修改条件所需的参数值,该数组中的值依次填充 whereClause 参数中的每一个问号。

代码段 9-18 是修改数据的一个示例,其功能是将 31 号学生的姓名和班级改为“李国刚”和“计算机 15”。

代码段 9-18 修改数据

```
SQLiteDatabase dbWriter=dbOpenHelper.getWritableDatabase();  
ContentValues cv =new ContentValues();  
cv.put("stuName","李国刚");  
cv.put("stuClass","计算机 15");  
dbWriter.update("student", cv, "stuID=?",new String[]{"31"});  
dbWriter.close();
```

数据表中数据的添加、删除和修改,除了分别使用上述 3 种方法以外,还可以通过调用 SQLiteDatabase 对象的 execSQL() 方法实现。execSQL() 方法可执行一条不返回结果的 SQL 语句。例如,向数据库的表中插入一行记录(20,'李庆华','计算机 15'),将记录(32,'刘凯旋','软件 152')修改为(32,'刘凯旋','计算机 152'),可通过执行代码段 9-19 中的语句实现。

代码段 9-19 插入和修改数据

```
SQLiteDatabase db=dbHelper.getWritableDatabase();  
//获取拥有修改权限的 SQLiteDatabase 实例对象  
db.execSQL("INSERT INTO student (stuId, stuName, stuClass) VALUES (20,'李庆华',  
    '计算机 15')");  
db.execSQL("UPDATE student SET stuClass='计算机 152' WHERE stuId= 32 AND stuName=  
    '刘凯旋';");
```

【例 9-8】 示例工程 Demo_09_WriteDatabase 演示了如何插入、删除、修改数据库中的数据。

运行结果如图 9-12 所示。在界面下方输入学号、姓名、班级,点击“添加数据”按钮,实现数据的添加;点击 ListView 中的某条数据,就会显示在下方的输入框中,修改数据后点击“修改数据”按钮,则会修改该条数据;长按 ListView 中的某条数据就会删除该条数

据,点击“全部删除”按钮,就会删除表中的全部数据。

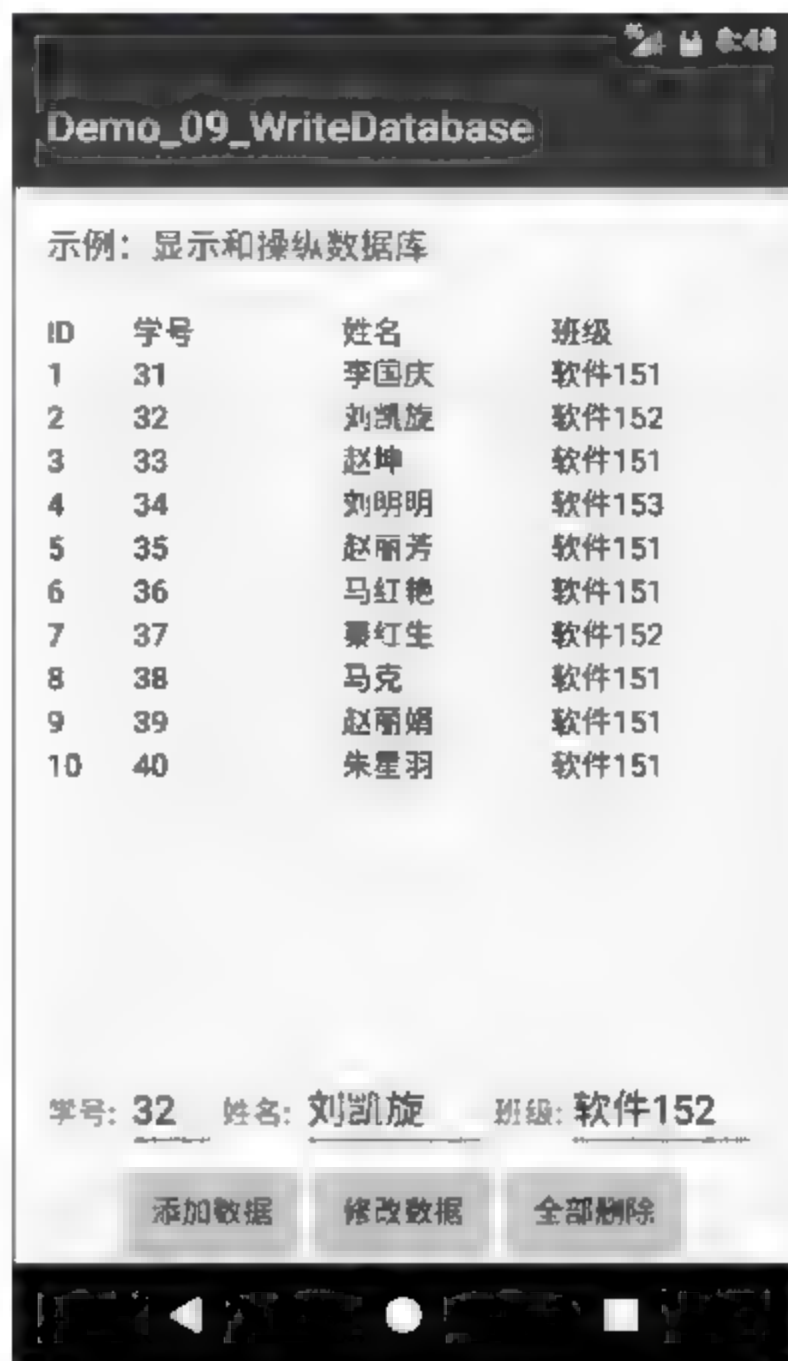


图 9-12 操纵数据库示例

程序代码如代码段 9-20 所示。

代码段 9-20 查询数据

//package 和 import 语句略

```
public class MainActivity extends AppCompatActivity {
    private SQLiteDatabase dbReader, dbWriter;
    private MyDBOpenHelper dbOpenHelper;
    private SimpleCursorAdapter listViewAdapter;
    private Button btnDataAdd, btnUpdate, btnDeleteAll;
    private EditText studentIdEdit, nameEdit, classEdit;
    private ListView listView;
    private String currentID = "1";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnDataAdd = (Button) findViewById(R.id.btnAdd);
        btnUpdate = (Button) findViewById(R.id.btnUpdate);
        btnDeleteAll = (Button) findViewById(R.id.btnDeleteAll);
        nameEdit = (EditText) findViewById(R.id.etStudentName);
        studentIdEdit = (EditText) findViewById(R.id.etStudentID);
    }
}
```

```
classEdit = (EditText) findViewById(R.id.etStudentClass);
listView = (ListView) findViewById(R.id.listView);
dbOpenHelper = new MyDBOpenHelper(getApplicationContext(),
    "SC Database.db", null, 1);
dbReader = dbOpenHelper.getReadableDatabase();
dbWriter = dbOpenHelper.getWritableDatabase();
showAll();
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?>parent, View view, int
        position, long id) {
        TextView itemID = (TextView) view.findViewById(R.id.item_id);
        TextView stuID = (TextView) view.findViewById(R.id.itemID);
        TextView stuName = (TextView) view.findViewById(R.id.itemName);
        TextView stuClass = (TextView) view.findViewById(R.id.itemClass);
        currentID = itemID.getText().toString();
        studentIdEdit.setText(stuID.getText().toString());
        nameEdit.setText(stuName.getText().toString());
        classEdit.setText(stuClass.getText().toString());
        Log.d("我的提示", "_id- - - "+currentID);
    }
});
listView.setOnItemLongClickListener(new AdapterView.
    OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?>parent, View view,
        int position, long id) {
        TextView itemID = (TextView) view.findViewById(R.id.item_id);
        currentID = itemID.getText().toString();
        dbWriter.delete("student", "_id= ?", new String[]{currentID});
        showAll();
        return true;
    }
});
btnDataAdd.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        //从 EditText 中获得相应的输入值
        SQLiteDatabase dbWriter = dbOpenHelper.getReadableDatabase();
        ContentValues cv = new ContentValues();
        cv.put("stuId", Integer.parseInt(studentIdEdit.getText().
            toString()));
        cv.put("stuName", nameEdit.getText().toString());
        cv.put("stuClass", classEdit.getText().toString());
        dbWriter.insert("student", null, cv);
    }
});
```

```

        //向数据表中添加数据,第二个参数是对空列的填充处理策略
        showAll();
    }
});
btnUpdate.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        //从 EditText 中获得相应的输入值
        SQLiteDatabase dbWriter = dbOpenHelper.getReadableDatabase();
        ContentValues cv = new ContentValues();
        cv.put("stuId", studentIdEdit.getText().toString());
        cv.put("stuName", nameEdit.getText().toString());
        cv.put("stuClass", classEdit.getText().toString());
        dbWriter.update("student", cv, "_id=?", new String[]{currentID});
        showAll();
    }
});
btnDeleteAll.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        dbWriter.delete("student", null, null);
        showAll();
    }
});
}
private void showAll() {
    Cursor result = dbReader.query("student", null, null, null, null, null,
        "stuId", null);
    if (!result.moveToFirst()) { //判断游标是否为空
        Toast.makeText(getApplicationContext(), "数据表中一个数据也没有!",
            Toast.LENGTH_LONG).show();
    }
    listViewAdapter = new SimpleCursorAdapter(getApplicationContext(), R.
        layout.item_list, result,
        new String[]{"_id", "stuId", "stuName", "stuClass"},
        new int[]{R.id.item_id, R.id.itemID, R.id.itemName, R.id.itemClass},
        CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);
    listView.setAdapter(listViewAdapter);
}
@Override
protected void onDestroy() {
    super.onDestroy();
    dbWriter.close();
    dbReader.close();
}
}

```

9.4 基于 ContentProvider 的数据存取

在 Android 中,通过文件和 SQLite 数据库可以存储数据,但是这些数据都是应用程序私有的,如果多个应用程序需要共享同样的数据,那么就需要使用 ContentProvider。

9.4.1 ContentProvider

ContentProvider 是 Android 的四大组件之一,它提供了应用程序间共享数据的机制和数据存储方式。系统为所有的 ContentProvider 建立了一个数据表 Data Model, Data Model 中保存了系统和用户共享的所有数据集,每个数据集就像数据库中的数据表一样,用 ID 区别每条数据,每一列代表每条数据的属性。每个 ContentProvider 对象都对外提供了一个公开的 Uri 来表示相应的数据集。

如果应用程序有数据需要共享时,就可以使用 ContentProvider 为这些数据定义一个 URI,其他的应用程序就可以通过这个 URI 来对数据进行操作。ContentProvider 使用的 URI 通常有两种形式:一种是指定全部数据,例如 `content://PhonesList/phones` 指的是全部通讯录数据;另一种是某个指定 ID 的数据,例如 `content://PhonesList/phones/1` 指的是 id 列值为 1 的通讯录数据。

所有的 URI 均由 3 部分组成:scheme、authority/host 和 path,它们的含义如下:

(1) scheme: 对于 ContentProvider,Android 规定的 scheme 为“content://”。

(2) authority/host: 授权者或主机名称,用于唯一标识一个 ContentProvider。外部应用程序可以根据这个标识来找到相应的共享数据。一般 authority 都由类的小写全称组成,以保证唯一性。

(3) path: 要操作的数据路径,用于确定请求的是哪一个数据集。

ContentProvider 与 Service、BroadcastReceiver 等组件一样,在 AndroidManifest.xml 配置文件里声明后调用者才可以使用。

9.4.2 定义和使用 ContentProvider

应用程序可以定义自己的 ContentProvider,其主要步骤如下:

步骤 1: 创建自己的数据存储,如数据库、文件或其他。

步骤 2: 创建一个继承于 ContentProvider 类的子类。在子类中重写 ContentProvider 的 6 个抽象方法 `query()`、`insert()`、`update()`、`delete()`、`getType()` 和 `onCreate()`,实现查询、插入、修改、删除数据、定义返回的 MIME 类型以及创建数据对象等操作接口。其中的前 4 个抽象方法分别对应于 ContentResolver 的 `query()`、`insert()`、`update()`、`delete()` 方法,当调用 ContentResolver 的这 4 个方法时,也就间接调用了 ContentProvider 的 4 个方法。

步骤 3: 在 AndroidManifest.xml 文件中声明新定义的 ContentProvider 及其对外共享标识 URI。

定义完成后,在其他应用程序中就可以对共享的 ContentProvider 数据进行操作。

应用程序可以通过 ContentResolver 接口存取 ContentProvider 共享数据。在 Activity 中,可以通过调用 getContentResolver() 方法得到当前应用的 ContentResolver 实例对象。ContentResolver 提供的接口和 ContentProvider 中需要实现的抽象方法对应,主要有 query()、insert()、update()、delete() 等,分别通过 URI 进行查询、插入、修改、删除。

【例 9-9】 示例工程 Demo_09_DBContentProvider 将数据库中的信息以 ContentProvider 的方式共享,这样做的好处是:如果以后另外一个程序需要访问此数据库中的数据时,只需要知道 ContentProvider 的 URI 即可。

具体方法是,创建 ContentProvider 的子类 MyContentProvider,重写 ContentProvider 类的抽象方法 query()、insert()、update()、delete()、getType() 和 onCreate(),如代码段 9-21 所示。ContentProvider 的增、删、改、查等操作实际上是间接调用了数据库的对应操作完成的。

代码段 9-21 定义 ContentProvider

```
//package 和 import 语句略
public class MyContentProvider extends ContentProvider {
    private MyDBOpenHelper dbHelper;
    public MyContentProvider() {
    }
    @Override
    public boolean onCreate() {
        //实例化 DBHelper 对象:
        dbHelper=new MyDBOpenHelper (getContext(),"SC_Database.db", null, 2);
        return true;
    }
    //获得数据库对象,并进行删除操作,返回删除的行数
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        SQLiteDatabase db =dbHelper.getWritableDatabase();
        return db.delete("tb_phones", selection, selectionArgs);
    }
    //获得数据库对象,并进行添加操作,返回添加最新行的 URI
    @Override
    public Uri insert(Uri uri, ContentValues values) {
        SQLiteDatabase db =dbHelper.getWritableDatabase();
        long i=db.insert("student",null,values);
        uri=ContentUris.withAppendedId(uri, i);
        return uri;
    }
}
```

```
//获得数据库对象,并进行查询操作,返回 Cursor 对象
@Override
public Cursor query(Uri uri, String[] projection, String selection,
    String[] selectionArgs, String sortOrder) {
    SQLiteDatabase db = dbOpenHelper.getWritableDatabase();
    Cursor c = db.query("student", projection, selection,
        selectionArgs, null, null, sortOrder);
    return c;
}
//获得数据库对象,并进行修改操作,返回修改的 row 值
@Override
public int update(Uri uri, ContentValues values, String selection, String[]
    selectionArgs) {
    SQLiteDatabase db = dbOpenHelper.getWritableDatabase();
    return db.update("student", values, selection, selectionArgs);
}
}
```

完成了 ContentProvider 子类的创建及方法重写后,需要在 AndroidManifest.xml 配置文件中声明这个 ContentProvider,如代码段 9-22 所示。

代码段 9-22 声明 ContentProvider

```
<provider
    android:name=".MyContentProvider"
    android:authorities="edu.hebust.xxy.demo_09_dbcontentprovider"
    android:enabled="true"
    android:exported="true">
</provider>
```

代码中的 name 值“. MyContentProvider”是程序中对应的 ContentProvider 的子类名称,authorities 的值就是这个 ContentProvider 对外公开的 URI。其他应用程序使用这些数据就是根据这个 URI 来找到它。

本例中 MainActivity 的界面布局如图 9-13 所示。

本例与前几个示例程序不同之处在于访问数据库的方式,不是使用 SQLiteDatabase 对象,而是使用 ContentResolver 对象,使用 URI 的方式访问 ContentProvider 中的共享数据。主要代码如代码段 9 23 所示。代码中,通过调用 getContentResolver()方法得到 ContentResolver 对象,然后调用该对象的 query()方法对 ContentProvider 进行查询,调用 insert()方法插入数据。

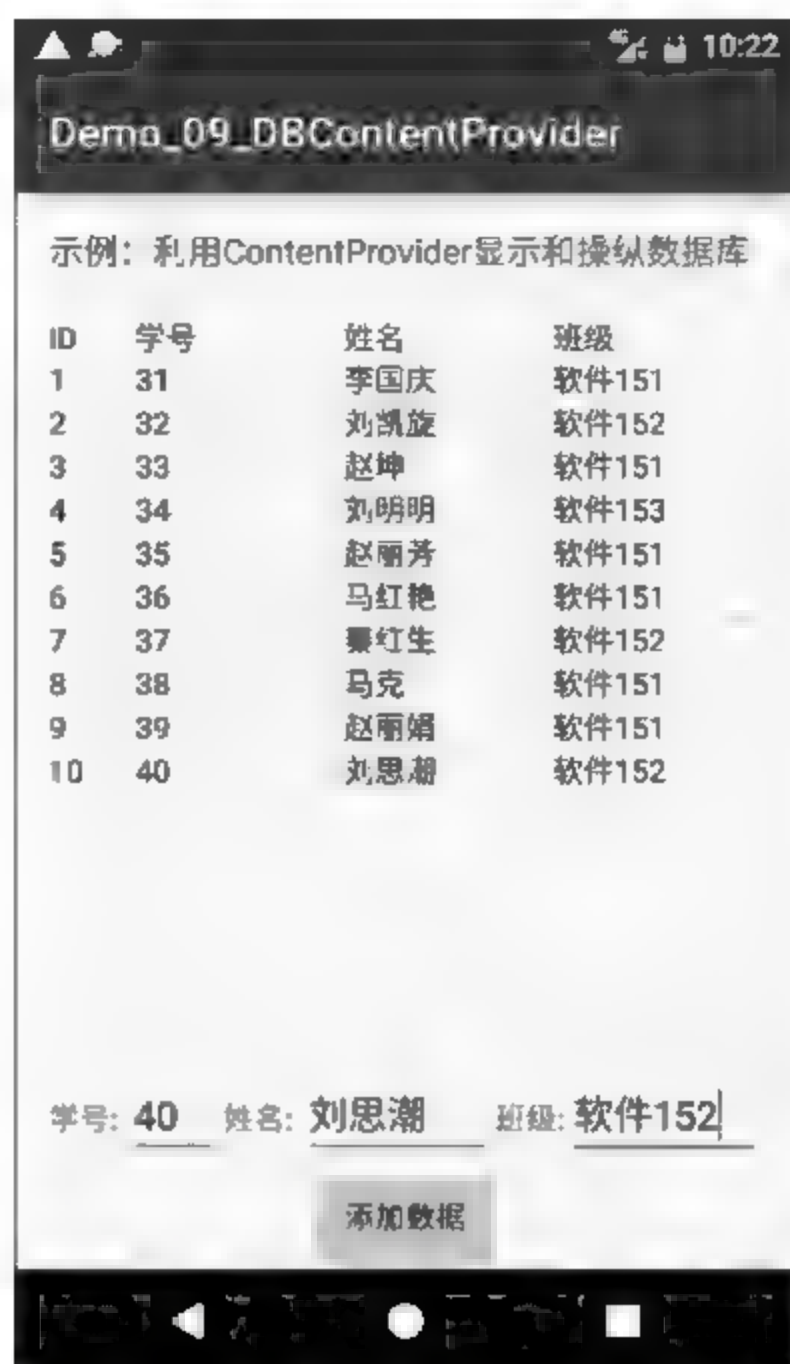


图 9-13 利用 ContentProvider 操纵数据库示例

代码段 9-23 使用 ContentProvider 访问数据库

```

public class MainActivity extends AppCompatActivity {
    private SimpleCursorAdapter listViewAdapter;
    private Button btnDataAdd;
    private EditText studentIdEdit, nameEdit, classEdit;
    private ListView listView;
    private String currentID="1";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnDataAdd = (Button) findViewById(R.id.btnAdd);
        nameEdit = (EditText) findViewById(R.id.etStudentName);
        studentIdEdit = (EditText) findViewById(R.id.etStudentID);
        classEdit = (EditText) findViewById(R.id.etStudentClass);
        listView = (ListView) findViewById(R.id.listView);
        showAll();
        btnDataAdd.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                ContentValues cv = new ContentValues();
                cv.put("stuId", Integer.parseInt(studentIdEdit.getText().
                    toString()));
            }
        });
    }
}

```

```
        cv.put("stuName", nameEdit.getText().toString());
        cv.put("stuClass", classEdit.getText().toString());
        String url="content://edu.hebust.xxy.demo_09_dbcontentprovider/
                student";
        getContentResolver().insert(Uri.parse(url), cv);
        showAll();
    }
});
}

private void showAll() {
    String url="content://edu.hebust.xxy.demo_09_dbcontentprovider/student";
    Cursor result =getContentResolver().query(Uri.parse(url),
        new String[]{"_id","stuId", "stuName", "stuClass"}, null,
        null,"stuId");
    if (!result.moveToFirst()) { //判断游标是否为空
        Toast.makeText(getApplicationContext(), "数据表中一个数据也没有!",
            Toast.LENGTH_LONG).show();
    }
    listViewAdapter =new SimpleCursorAdapter(getApplicationContext(),
        R.layout.item_list, result,
        new String[]{"_id","stuId", "stuName", "stuClass"},
        new int[]{R.id.item_id,R.id.itemID, R.id.itemName, R.id.itemClass},
        CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER);
    listView.setAdapter(listViewAdapter);
}
}
```

Android 系统为常见的一些数据提供了 ContentProvider,包括音频、视频、图片和联系人等,每个 ContentProvider 都会对外提供一个包装成 Uri 对象的公共 URI。这些 ContentProvider 称为系统 ContentProvider。系统 ContentProvider 和自定义的 ContentProvider 在使用上并没有什么区别,只不过系统 ContentProvider 在使用时需要注册其使用权限并知道它的 URI,而自定义的则不需要,只需要知道它的 URI 即可。例如要读取手机联系人的数据,那么就必须在 AndroidManifest.xml 文件中注册相应的 READ_CONTACTS 使用权限,即在 AndroidManifest.xml 文件中加入下面的语句:

```
<uses-permissionandroid:name="android.permission.READ_CONTACTS"/>
```

9.5 本章小结

本章介绍了在 Android 中如何实现数据的存储和获取,以及如何在应用程序之间利用 ContentProvider 共享数据存储。其中 SharedPreferences 是一种轻量级的数据存储机

制,以键-值对的方式将数据存储存储在 XML 文件中,适用于存储应用程序的配置信息。而文件和 SQLite 数据库都可以存储大容量的数据,它们具有不同的存储机制和操作方法。学习本章要熟练掌握各种数据存取方法,并能在应用程序设计中灵活运用这些方法。

习 题

1. Android 系统提供了哪些数据存储和访问方式?
2. 设计一个应用程序,界面中有一个 TextView,其中显示有若干文字。为 Activity 添加菜单,包括“红”“绿”“蓝”3 个菜单项,用户选择一个菜单项,即将 TextView 中的文字设为相应的颜色,同时将颜色信息写入到 SharedPreferences 中,下一次启动该程序时,文字的颜色默认为上一次关闭程序时文字的颜色。
3. 设计一个用于注册的 Activity。要求界面中的注册项包括用户名、账号、密码、性别、出生年月日、爱好。界面中有一个“注册”按钮,用户点击“注册”按钮后,将注册信息写入到 SharedPreferences,写入完成后将 SharedPreferences 信息读出并回显到 Activity 中。
4. 将第 3 题的注册信息写入到应用程序的私有文件,文件中每行存储一项注册信息,文件名为 count.txt,写入完成后将文件中的信息读出并回显到 Activity 中。
5. 将第 3 题的注册信息写入到应用程序的私有数据库中,数据表名称为 users,写入完成后将数据库中的信息读出并回显到 Activity 中。
6. 编写一个程序,继承 SQLiteOpenHelper 实现下述功能:创建一个版本为 1 的 diary.db 数据库,同时创建一个 diary 表,包含一个_id 主键并自增长,topic 字段(字符型,最大长度为 100 个字符),content 字段(字符型,最大长度为 1000 个字符),在数据库版本变化时删除 diary 表,并重新创建 diary 表。
7. 设计一个利用 SQLite 数据库存储和操纵数据的应用程序,创建一个商品基本信息表(product),包含商品编号、名称、价格、描述 4 个字段,实现表数据的增、删、改、查。
8. 简述 ContentProvider 是如何实现数据共享的,并尝试设计一个属于自己的 ContentProvider。
9. 在 AndroidManifest.xml 配置文件中声明 ContentProvider 的目的是什么?如何进行声明?



在智能移动设备的应用中,音视频等多媒体应用是一个重要的方面。本章将介绍在 Android 系统中如何处理和使用音视频和照片等资源,包括音频和视频的播放及录制、照片的摄取等。

10.1 音视频文件的播放

Android 系统提供了对常用音视频文件格式的支持,包括 MP3、WAV、OGG 等音频格式和 MP4、3GPP 等视频格式。通过 Android API 提供的相关方法,可以实现音视频文件的播放。

10.1.1 MediaPlayer 类

`android.media.MediaPlayer` 类用于播放音视频文件,提供了对音视频操作的一些重要方法,如播放、停止、暂停、重复播放等。播放的音视频文件可以来自 RAW 源文件、本地文件系统和通过网络传送的文件流。

`MediaPlayer` 的运行是基于状态的。当一个 `MediaPlayer` 对象被刚刚用 `new` 操作符创建或是调用了 `reset()` 方法后,它就处于 `Idle`(空闲)状态。当调用了 `release()` 方法后,它就处于 `End`(结束)状态。这两种状态之间是 `MediaPlayer` 对象的生命周期。`MediaPlayer` 的状态及其转换如图 10-1 所示。

1. Idle(空闲)状态

使用 `new` 方法创建一个新的 `MediaPlayer` 对象,或调用 `reset()` 方法,使一个 `MediaPlayer` 对象处于 `Idle` 状态。

2. Initialized(已初始化)状态

调用 `setDataSource(FileDescriptor)`、`setDataSource(String)`、`setDataSource(Context, Uri)`,或 `setDataSource(FileDescriptor,long,long)` 方法会使处于 `Idle` 状态的对象迁移到 `Initialized` 状态。

注意: 若当 `MediaPlayer` 对象处于其他的状态(非 `Idle` 状态)下,调用 `setDataSource()` 方

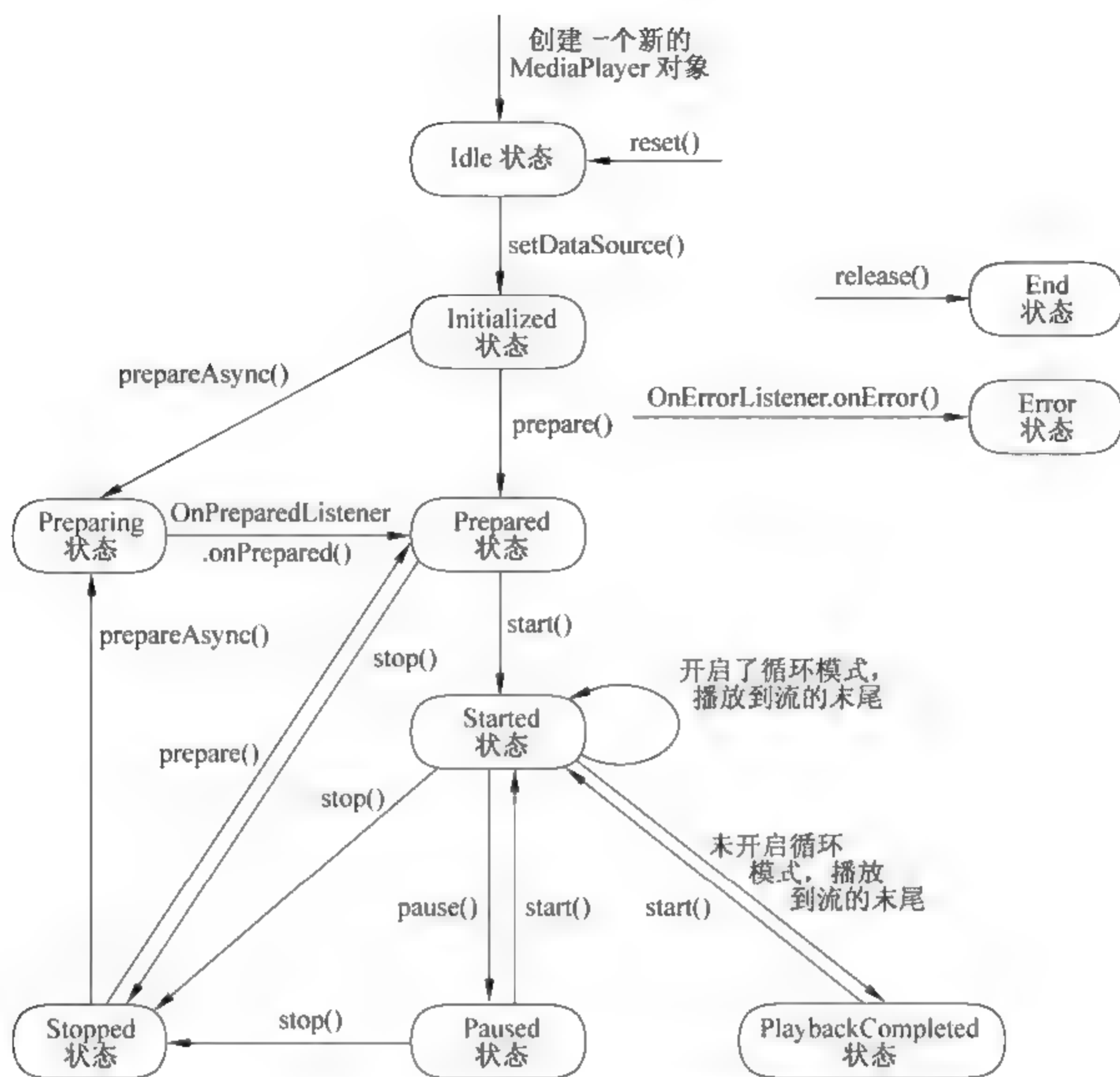


图 10-1 MediaPlayer 的生命周期示意图

法,会抛出 `IllegalStateException` 异常。

3. Prepared/Preparing(就绪/准备)状态

在开始播放之前,MediaPlayer 对象必须进入就绪状态。

有两种方法(同步和异步)可以使 MediaPlayer 对象进入就绪状态:调用 `prepare()` 方法(同步),使该 MediaPlayer 对象进入 Prepared 状态并返回;或调用 `prepareAsync()` 方法(异步),使 MediaPlayer 对象进入 Preparing 状态并返回,而内部的播放引擎会继续未完成的准备工作。当同步版本返回时或异步版本的准备工作全部完成时,就会调用客户端提供的 `OnPreparedListener.onPrepared()` 监听方法。可以调用 `MediaPlayer.setOnPreparedListener(android.media.MediaPlayer.OnPreparedListener)` 方法来注册 `OnPreparedListener`。

在不合适的状态下调用 `prepare()` 和 `prepareAsync()` 方法会抛出 `IllegalStateException` 异常。当 MediaPlayer 对象处于 Prepared 状态的时候,可以调整音视频的属性,如音量、播放时是否一直亮屏、循环播放等。

4. Started(播放)状态

要开始播放,必须调用 `start()` 方法。当此方法成功返回时, `MediaPlayer` 的对象处于 `Started` 状态。可以调用 `isPlaying()` 方法来测试某个 `MediaPlayer` 对象是否在 `Started` 状态。

当处于 `Started` 状态时,内部播放引擎会调用客户端程序提供的 `OnBufferingUpdateListener.onBufferingUpdate()` 回调方法,此回调方法允许应用程序追踪播放流缓冲的状态。

对一个已经处于 `Started` 状态的 `MediaPlayer` 对象调用 `start()` 方法将不会执行任何操作。

5. Paused(暂停)状态

调用 `pause()` 方法可以暂停、停止播放,以及调整当前播放位置。当调用 `pause()` 方法并返回时,会使 `MediaPlayer` 对象进入 `Paused` 状态。`Started` 与 `Paused` 状态的相互转换在内部的播放引擎中是异步的,所以可能需要一点时间在 `isPlaying()` 方法中更新状态,若正在播放流内容,这段时间可能会有几秒。

调用 `start()` 方法会让一个处于 `Paused` 状态的 `MediaPlayer` 对象从之前暂停的地方恢复播放。当调用 `start()` 方法返回的时候, `MediaPlayer` 对象的状态会又变成 `Started` 状态。

对一个已经处于 `Paused` 状态的 `MediaPlayer` 对象调用 `pause()` 方法将不会执行任何操作。

6. Stopped(停止)状态

调用 `stop()` 方法会停止播放,并且还会让一个处于 `Started`、`Paused`、`Prepared` 或 `PlaybackCompleted` 状态的 `MediaPlayer` 进入 `Stopped` 状态。

对一个已经处于 `Stopped` 状态的 `MediaPlayer` 对象调用 `stop()` 方法将不会执行任何操作。

调用 `seekTo()` 方法可以调整播放的位置。`seekTo(int)` 方法是异步执行的,所以它可以马上返回,但是实际的定位播放操作可能需要一段时间才能完成,尤其是播放流形式的音视频。`seekTo(int)` 方法可以在其他状态下调用,例如 `Prepared`、`Paused` 和 `PlaybackCompleted` 状态。此外,当前的播放位置可以通过调用 `getCurrentPosition()` 方法得到,它可以用于帮助播放应用程序不断更新播放进度。

7. PlaybackCompleted(播放完成)状态

当播放到流的末尾,播放就完成了。如果调用了 `setLooping(boolean)` 方法开启了循环模式,那么这个 `MediaPlayer` 对象会重新进入 `Started` 状态。如果没有开启循环模式,那么内部的播放引擎会调用客户端程序提供的 `OnCompletion.onCompletion()` 回调方法。可以通过调用 `MediaPlayer.setOnCompletionListener(OnCompletionListener)` 方法

来设置。内部的播放引擎一旦调用了 `OnCompletion.onCompletion()` 回调方法,说明这个 `MediaPlayer` 对象进入了 `PlaybackCompleted` 状态。

当处于 `PlaybackCompleted` 状态的时候,可以再调用 `start()` 方法让这个 `MediaPlayer` 对象再进入 `Started` 状态。

8. Error(错误)状态

一旦发生错误,`MediaPlayer` 对象会进入 `Error` 状态。可以调用 `reset()` 方法把这个对象恢复成 `Idle` 状态。在不合法的状态下调用一些方法,如 `prepare()`、`prepareAsync()` 或 `setDataSource()` 方法会抛出 `IllegalStateException` 异常,使 `MediaPlayer` 对象进入 `Error` 状态。

9. End(结束)状态

当调用了 `release()` 方法后,`MediaPlayer` 对象就处于 `End` 状态。一旦一个 `MediaPlayer` 对象不再被使用,应立即调用 `release()` 方法来释放在内部播放引擎中与这个 `MediaPlayer` 对象关联的资源。资源可能包括硬件加速组件的单态组件,若没有调用 `release()` 方法可能会导致之后的 `MediaPlayer` 对象实例无法使用这种单态硬件资源,导致运行失败。一旦 `MediaPlayer` 对象进入了 `End` 状态,就不能再被使用,也没有办法再迁移到其他状态。

特定的操作只能在特定的状态时才有效,所以编写程序时必须时刻注意到它的变化。如果在错误的状态下执行一个操作,系统可能抛出一个异常或导致一个意外的行为。例如,当创建一个新的 `MediaPlayer` 对象时,它处于 `Idle` 状态,应调用 `setDataSource()` 方法初始化,使它进入 `Initialized` 状态。之后,应调用 `prepare()` 方法或 `prepareAsync()` 方法准备播放。当 `MediaPlayer` 准备完成,它将进入 `Prepared` 状态,这表示可以调用 `start()` 来播放了。注意,当调用了 `stop()` 方法后,不能再调用 `start()` 方法,除非使其重新进入 `Prepared` 状态。

10.1.2 使用 MediaPlayer 播放音频文件

可以利用 `MediaPlayer` 对象来播放音频文件。在使用 `MediaPlayer` 之前,必须在 `AndroidManifest.xml` 配置文件中声明相关的权限。如果播放的文件在外部存储中,需要在 `AndroidManifest.xml` 配置文件中声明外存的读权限,权限名称为 `android.permission.READ_EXTERNAL_STORAGE`;如果应用程序在播放过程中需要阻止屏幕变暗或阻止处理器睡眠,或使用 `MediaPlayer.setScreenOnWhilePlaying()`、`MediaPlayer.setWakeMode()` 等方法,还必须声明对睡眠加锁的权限,权限名称为 `android.permission.WAKE_LOCK`;如果使用 `MediaPlayer` 来播放网络流中的内容,还必须声明网络存取权限,权限名称为 `android.permission.INTERNET`。

实现播放需要首先创建 `MediaPlayer` 对象,并装载音频文件,然后调用 `MediaPlayer` 对象的 `start()` 方法播放音频文件。

(1) 创建 `MediaPlayer` 对象,装载音频文件。

可以通过调用 MediaPlayer 的静态方法 create() 创建 MediaPlayer 对象,也可以通过它的构造方法用 new 操作符创建 MediaPlayer 对象。

create() 方法的语法格式如下:

```
public static MediaPlayer create(Context context, int resid)
```

或

```
public static MediaPlayer create(Context context, Uri uri)
```

代码段 10-1 分别用两种方法创建了 MediaPlayer 对象 mpRaw 和 mpLocal,并且分别装载了音频文件。

代码段 10-1 创建 MediaPlayer 对象,装载音频文件

```
MediaPlayer mpRaw = MediaPlayer.create(this, R.raw.musicname);  
//创建 MediaPlayer 对象 mpRaw 并装载 raw 文件夹中 musicname.mp3 音频文件  
MediaPlayer mpLocal = new MediaPlayer();  
//创建 MediaPlayer 对象 mpLocal  
mpLocal.setDataSource("/storage/emulated/0/music/music01.mp3");  
//装载本地文件系统存储的音频文件 music01.mp3
```

MediaPlayer 播放的音频文件可以是 RAW 资源文件、本地文件系统或网络中的音频文件。Android 系统不会解析 RAW 资源文件,它必须是一种适当编码和格式化的媒体文件。

MediaPlayer 播放网络音频文件的具体方法是,通过创建网络 URI 实例,调用 MediaPlayer 的静态方法 create(),传递 URI 参数来完成 MediaPlayer 对象实例化,或通过调用 MediaPlayer 对象的 setDataSource() 方法,设置文件播放路径来完成播放。

需要注意的是,当使用 setDataSource() 方法时必须捕获和传递 IllegalArgumentException 和 IOException,因为引用的文件可能不存在。使用 setDataSource() 方法设置要装载的音频文件后,MediaPlayer 并没有真正装载这个文件,因此需要调用 MediaPlayer 的 prepare() 方法装载这个文件,之后才能播放。

(2) 调用 MediaPlayer 对象的 start() 方法播放音频文件。

设置了 MediaPlayer 对象的数据源后,可以先调用 MediaPlayer 对象的 prepare() 方法进行播放前的准备,之后调用 start() 方法播放指定的多媒体音频文件。

(3) 播放过程的控制。

如果想停止音频文件的播放,可以调用 MediaPlayer 对象的 stop() 方法停止播放;暂停播放调用 pause() 方法;重复播放则需要先调用 reset() 方法初始化 MediaPlayer 状态,然后调用 prepare() 方法准备播放,最后调用 start() 方法播放媒体文件。当暂停文件播放后,如果要继续播放,重新调用 start() 方法即可。

【例 10 1】 示例工程 Demo 10 MediaPlayerForAudio 实现了一个简易的音频文件播放器,播放 raw 文件夹下的音频文件。播放过程中可以进行暂停、继续、停止的控制。

MainActivity 的布局如图 10-2 所示,程序运行后会自动播放音频文件。



图 10-2 简易播放器的界面

点击“暂停”按钮,调用 MediaPlayer 对象的 pause() 方法;点击“播放”按钮,调用 start() 方法继续播放;点击“停止”按钮,则调用 stop() 方法使播放停止。具体实现如代码段 10-2 所示。

代码段 10-2 简易的音频文件播放器程序

//package 和 import 语句略

```
public class MainActivity extends AppCompatActivity {
    MediaPlayer mp;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView text = (TextView) this.findViewById(R.id.text);
        Button BtnPau = (Button) this.findViewById(R.id.BtnPau);
        Button BtnCon = (Button) this.findViewById(R.id.BtnCon);
        Button BtnStop = (Button) this.findViewById(R.id.BtnStop);
        text.setText("正在播放 Raw 资源文件 music01.mp3");
        mp=MediaPlayer.create(this, R.raw.music01);
        //创建 MediaPlayer 对象 mp 并关联音频文件
        mp.start(); //播放音频文件
        BtnPau.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mp.pause();
                text.setText("播放暂停");
            }
        });
        BtnCon.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mp.start();
            }
        });
    }
}
```

```
        text.setText("正在播放 Raw 资源文件 music01.mp3");
    }
});
BtnStop.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mp.stop();
        text.setText("播放停止");
    }
});
}
}
```

10.1.3 使用 MediaPlayer 播放视频文件

使用 MediaPlayer 类播放视频文件,需要使用一个 SurfaceView 对象作为输出设备。

SurfaceView 继承自 android.view.View 类,视图中内嵌了一个专门用于绘制的 Surface,可以控制这个 Surface 的格式、尺寸以及绘制位置。SurfaceView 通常与 MediaPlayer 结合使用,提供一个播放视频的预览窗口。

为了节省资源,SurfaceView 变得可见时内嵌的 Surface 被创建,SurfaceView 隐藏前 Surface 被销毁。可以通过 SurfaceHolder 接口访问这个 Surface,调用 getHolder() 方法可以得到这个接口。Surface 是纵深排序(Z ordered)的,它总在自己所在窗口的后面。SurfaceView 提供了一个可见区域,只有在这个可见区域内的 Surface 部分内容才可见,可见区域外的部分不可见。Surface 的排序显示受到视图层级关系的影响,它的兄弟视图结点会在顶端显示。这意味着 Surface 的内容会被它的兄弟视图遮挡,这一特性可以用来放置遮盖物(overlays),例如文本和按钮等控件。但是要注意,如果 Surface 上面有透明控件,那么它的每次变化都会引起框架重新计算它和顶层控件的透明效果,这会影响性能。

SurfaceView 默认使用双缓冲技术,它支持在子线程(渲染线程)中绘制图像,这样就不会阻塞主线程了,所以它非常适合游戏的开发。一般来说,所有 SurfaceView 和 SurfaceHolder.Callback 的方法都应该在 UI 线程里调用,子线程所要访问的各种变量应该作同步处理。由于 Surface 可能被销毁,它只在 SurfaceHolder.Callback.surfaceCreated() 和 SurfaceHolder.Callback.surfaceDestroyed() 之间有效,所以要确保渲染线程访问的是合法有效的 surface。

初始的 SurfaceView 将决定视频的播放大小。系统会自动适配 SurfaceView 和视频的大小比例,使之恰当。

使用 SurfaceView 的一般过程是:首先继承 SurfaceView 并实现 SurfaceHolder.Callback 接口,实现它的 3 个方法是 surfaceCreated()、surfaceChanged() 和 surfaceDestroyed()。还需要获得 SurfaceHolder,并添加回调函数,这样这 3 个方法才会执行。

surfaceCreated(SurfaceHolder holder)方法在 surface 创建的时候调用,一般在该方法中启动绘图的线程。surfaceChanged(SurfaceHolder holder, int format, int width, int height)方法在 surface 尺寸发生改变的时候调用,如横竖屏切换。surfaceDestroyed(SurfaceHolder holder)方法在 Surface 被销毁的时候调用,一般在该方法中停止绘图线程。

【例 10-2】 示例工程 Demo_10_MediaPlayerForVideo 演示了使用 MediaPlayer 和 SurfaceView 播放视频的方法。

本例中播放外部存储中的视频文件,所以需要在 AndroidManifest.xml 配置文件中注册外存的读权限:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

在界面布局文件中添加 SurfaceView 控件,布局文件内容如代码段 10-3 所示。

代码段 10-3 在界面布局文件中添加 SurfaceView 控件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="使用 MediaPlayer 播放视频示例"/>
    <TextView
        android:id="@+id/text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
    <SurfaceView
        android:id="@+id/myVideoView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```

定义 MainActivity 类,获取布局文件中的 SurfaceView 实例,并实现 SurfaceHolder.Callback 接口,重写它的 surfaceCreated()方法,实现视频的播放,如代码段 10-4 所示。

代码段 10-4 视频的播放

```
//package 和 import 语句略
public class MainActivity extends AppCompatActivity {
    private SurfaceView mySurfaceView;
    private SurfaceHolder myHolder = null;
```

```
private MediaPlayer myMediaPlayer;
private String myPath;
private TextView text;
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    this setContentView(R.layout.activity_main);
    text = (TextView) this.findViewById(R.id.text);
    mySurfaceView = (SurfaceView) this.findViewById(R.id.myVideoView);
    myPath = Environment.getExternalStorageDirectory().getPath() +
        "/movies/video0010.3gp";
    //SurfaceView 中的 getHolder 方法可以获取一个 SurfaceHolder 实例
    myHolder = mySurfaceView.getHolder();
    myHolder.addCallback(new SurfaceHolder.Callback() {
        @Override
        public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        }
        @Override
        public void surfaceCreated(SurfaceHolder holder) {
            //当 Surface 被创建时,该方法被调用,可以在这里实例化 Camera 对象
            try {
                myMediaPlayer = new MediaPlayer();
                myMediaPlayer.setDataSource(myPath);
                text.setText("播放视频文件:" + myPath);
                myMediaPlayer.setDisplay(holder);
                myMediaPlayer.prepare();
                myMediaPlayer.start();
            } catch (Exception e) {
                Log.e("我的提示信息", "error: " + e.getMessage(), e);
            }
        }
        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {
            //当 Surface 被销毁的时候,该方法被调用
        }
    });
}
```

示例程序的运行结果如图 10-3 所示。

10.1.4 利用系统内置的播放器程序播放音频和视频

对于音频和视频,Android 系统有其内置的播放器程序,可以使用隐式 Intent 来调用它。



图 10-3 利用 MediaPlayer 播放视频文件

使用 Android 内置的播放器,只需指定 Intent 对象的 Action 属性值为 ACTION_VIEW,同时用一个 URI 来指定要播放文件的路径,并指定所要播放文件的格式信息(MIME),就可以调用播放器来播放该音频或视频了。

【例 10 3】 示例工程 Demo_10_MusicPlay 演示了如何利用 Android 内置的播放器来播放音频文件。

MainActivity 类的主要代码如代码段 10 5 所示,示例程序的运行结果如图 10 4 所示。

代码段 10-5 使用 Android 系统内置的播放器程序播放音频

//package 和 import 语句略

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Intent intent = new Intent(Intent.ACTION_VIEW);  
        File sdcard = Environment.getExternalStorageDirectory();  
        //获取外部存储的路径  
        File audioFile = new File(sdcard.getPath() + "/music/music01.mp3");  
        //获取音频文件的 Uri
```

```
Uri audioUri = Uri.fromFile(audioFile);  
//指定 Uri 和 MIME  
intent.setDataAndType(audioUri, "audio/x-mpeg");  
startActivity(intent);           //播放  
}  
}
```

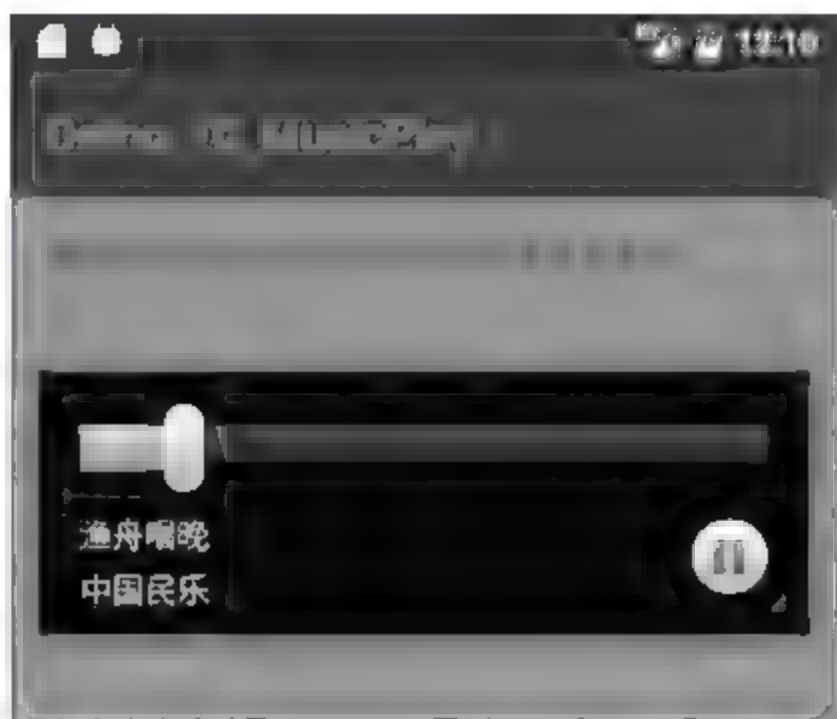


图 10-4 调用系统内置的播放器播放音频

【例 10 4】 示例工程 Demo_10_VideoPlay 演示了使用系统内置播放器播放视频的方法。

MainActivity 类的主要代码如代码段 10-6 所示。

代码段 10-6 使用 Android 系统内置的播放器程序播放视频

//package 和 import 语句略

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Intent intent = new Intent(Intent.ACTION_VIEW); //设置 Action  
        File sdcard = Environment.getExternalStorageDirectory();  
                                                    //获取外部存储的路径  
        //获取该文件的 URI  
        Uri uri = Uri.parse(sdcard.getPath() + "/movies/video0010.3gp");  
        //设置视频文件的类型  
        intent.setDataAndType(uri, "video/3gp");  
        startActivity(intent); //调用系统内置的播放器,开始播放  
    }  
}
```

示例程序运行后,调用 Android 内置的播放器全屏播放指定的视频文件,如图 10-5 所示。播放完成后自动回到 MainActivity 的界面。



图 10-5 调用系统内置的播放器播放视频

10.1.5 使用 VideoView 播放视频

为了简化播放视频文件的处理过程,Android 框架提供了 VideoView 类来封装 MediaPlayer。VideoView 在 android.widget 包中,继承自 android.view.SurfaceView 类,实现了 MediaController、MediaPlayerControl 接口,用于播放视频和播放过程的控制。VideoView 通过与 MediaController 类结合使用,编程者可以不用自己控制播放与暂停,它的使用过程比利用 SurfaceView 结合 MediaPlayer 播放视频更直接、简单。

VideoView 类可以从资源文件或内容提供者等不同的来源读取视频图像,并能计算和维护视频的画面尺寸,以使其适用于任何布局管理器。另外,它还提供了一些诸如缩放、着色之类的显示选项。

VideoView 的主要构造方法如下:

(1) `public VideoView (Context context)`。

该方法创建一个默认属性的 VideoView 实例。参数 context 是视图运行的应用程序上下文,通过它可以访问当前主题、资源等。

(2) `public VideoView (Context context, AttributeSet attrs)`。

该方法创建一个带有 attrs 属性的 VideoView 实例。参数 context 是视图运行的应用程序上下文,attrs 是用于视图的 XML 标签属性集合。

(3) `public VideoView (Context context, AttributeSet attrs, int defStyle)`。

该方法创建一个带有 attrs 属性,并且指定其默认样式的 VideoView 实例。参数 defStyle 是应用到视图的默认风格。如果为 0 则不应用风格(包括当前主题中的风格)。

VideoView 提供了一些公共方法用于播放过程的控制,例如,调用 `start()` 方法开始播放视频文件,调用 `pause()` 方法使播放暂停,调用 `seekTo()` 方法设置播放位置,调用 `setVideoPath()` 和 `setVideoURI()` 方法设置视频文件的路径,等等。VideoView 还提供了一些方法用于获得播放过程各类参数,例如,调用 `getCurrentPosition()` 方法可以获得

当前的播放位置,调用 `getDuration()` 方法可以获得所播放视频的总时间,调用 `isPlaying()` 方法可以判断是否正在播放视频等。

当 `VideoView` 创建的时候, `MediaPalyer` 对象将会创建;当 `VideoView` 对象销毁的时候, `MediaPlayer` 对象将会释放。不需要管理 `MediaPalyer` 的各种状态,这些状态都已经被 `VideoView` 封装了。

【例 10-5】 工程 Demo 10 `VideoView` 使用 `VideoView` 实现播放视频。在布局文件中使用 `VideoView` 结合 `MediaController` 来实现对视频播放的控制。

首先,在界面布局文件中添加 `VideoView` 控件,或在 Java 程序中创建 `VideoView` 对象。本例使用前一种方法, `activity_main` 文件内容如代码段 10-7 所示。

代码段 10-7 `activity_main.xml` 文件中添加 `VideoView` 控件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="用 VideoView 播放视频文件示例"/>
    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <VideoView
        android:id="@+id/video_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_centerInParent="true" />
</LinearLayout>
```

然后,在 `MainActivity` 中获取布局文件中的 `VideoView` 实例,调用 `VideoView` 类的方法加载指定的视频文件。有两个方法可以实现这一功能,其中 `setVideoPath(String path)` 方法用于加载 `path` 路径指定的视频文件,而 `setVideoURI(Uri uri)` 方法用于加载 `URI` 所对应的视频文件,可视具体情况选择其一。调用 `VideoView` 的 `start()`、`stop()`、`pause()` 方法控制视频的播放。`MainActivity` 的主要代码如代码段 10-8 所示。

代码段 10-8 使用 `VideoView` 播放视频

```
//package 和 import 语句略
public class MainActivity extends AppCompatActivity implements MediaPlayer.
    OnErrorListener, MediaPlayer.OnCompletionListener{
```

```

private VideoView myVideoView;
private Uri mUri;
private int mPositionWhenPaused = 1;
private MediaController mMediaController;
private TextView textView;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    myVideoView = (VideoView)findViewById(R.id.video_view);
    textView = (TextView)findViewById(R.id.text_view);
    File sdcard = Environment.getExternalStorageDirectory();
                                                    //获取外部存储路径
    mUri = Uri.parse(sdcard.getPath()+"/movies/video0010.3gp");
                                                    //将路径字符串解析为 URI 实例
    textView.setText("播放视频:"+mUri.toString()+"\n");
    mMediaController = new MediaController(this);
                                                    //创建媒体控制器
    myVideoView.setMediaController(mMediaController);
                                                    //设置一个控制条
}
@Override
public void onStart() {
    super.onStart();
    myVideoView.setVideoURI(mUri);
                                                    //设置视频文件的路径
    myVideoView.start();
                                                    //播放视频
}
//监听 MediaPlayer 上报的错误信息
public boolean onError(MediaPlayer mp, int what, int extra) {
    return false;
}
//Video 播完的时候得到通知
public void onCompletion(MediaPlayer mp) {
    this.finish();
}
}

```

通过实现 MediaPlayer.OnErrorListener 接口可以监听 MediaPlayer 上报的错误信息。另外,实现 MediaPlayer.OnCompletionListener 接口,将会在 Video 播完的时候得到通知,本例只是设置了在播完后结束程序。

示例程序为 VideoView 设置了一个 Android 内置的控制条,具有“暂停”“快进”“快退”按钮和一个进度显示条。示例程序的运行结果如图 10-6 所示。



图 10-6 利用 VideoView 播放视频文件

10.2 音视频文件的录制

10.2.1 MediaRecorder 类

MediaRecorder 类用于录制音频和视频文件。与 MediaPlayer 类似,它的运行也是基于状态的。MediaRecorder 主要有以下几个状态:

(1) Initial(初始)状态。使用 new 方法创建一个新的 MediaRecorder 对象,则它处于 Initial 状态。

(2) Initialized(已初始化)状态。MediaRecorder 对象在 Initial 状态时,设定视频源或音频源之后将转换为 Initialized 状态。通过调用 reset() 方法可以回到 Initial 状态。

(3) DataSourceConfigured(数据源配置)状态。MediaRecorder 对象在 Initialized 状态时,可以通过设置输出格式转换为 DataSourceConfigured 状态。这个期间可以设定编码方式、输出文件、屏幕旋转、预览显示等。它仍然可以通过调用 reset() 方法回到 Initial 状态。

(4) Prepared(就绪)状态。MediaRecorder 对象在 DataSourceConfigured 状态时,可以通过调用 prepare() 方法进入 Prepared 状态。通过调用 reset() 方法回到 Initialized 状态。

(5) Recording(录制)状态。MediaRecorder 对象在 Prepared 状态下,通过调用 start()

方法可以进入 Recording 状态。它可以通过停止或者重新启动回到 Initial 状态。

(6) Released(释放)状态。可以通过调用 `release()` 方法进入这个状态,这时将会释放所有和 `MediaRecorder` 对象绑定的资源。

(7) Error(错误)状态。当错误发生的时候进入 Error 状态,可以调用 `reset()` 方法把这个对象恢复成 Initial 状态。

10.2.2 使用 MediaRecorder 录制音视频

使用 `MediaRecorder` 类可以实现音视频的录制功能。使用 `MediaRecorder` 类录制音频和视频的方法类似,所不同的是录制视频需要设置更多的参数,例如设置用来录制视频的 `Camera`、视频图像的输出尺寸、视频的编码格式等。另外,视频录制需要使用 `Camera`,还需要在 `AndroidManifest.xml` 配置文件中声明相关权限。

使用 `MediaRecorder` 录制音频和视频的一般步骤如下。

步骤 1: 在 `AndroidManifest.xml` 配置文件中声明相关权限。录制音频需要获得录制 `audio` 的权限,视频录制需要获得 `Camera` 的权限,如果要将录制的文件写入外部存储中,则还需要外存的写入权限。

步骤 2: 定义 `MainActivity` 类,创建 `MediaRecorder` 实例。可以用 `MediaRecorder` 的默认构造方法创建一个 `MediaRecorder` 的实例对象。

步骤 3: 设置 `MediaRecorder` 对象的数据源。音频录制需要调用 `MediaRecorder` 对象的 `setAudioSource()` 方法设置音频源。例如下面的语句指定音频源为 `MIC`,即从麦克风获取音频,这是最常用的音频源。

```
myRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
```

类似地,如果进行视频录制,需要调用 `MediaRecorder.setVideoSource()` 方法设置视频源。例如下面的语句指定从照相机采集视频。

```
myRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
```

步骤 4: 设置音视频的输出格式和编码格式。例如下面的代码片段指定了音频编码方式为 `AMR_NB`,视频编码格式为 `H263` 格式。

```
myRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
myRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
myRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.H263);
```

如果是录制视频,还需要设置视频的分辨率和视频帧率,例如:

```
myRecorder.setVideoSize(960, 544);
myRecorder.setVideoFrameRate(4);
```

MediaRecorder.OutputFormat.THREE_GPP 为 MediaRecorder 音视频输出格式的一种,其他的还有 AMR_NB、AMR_WB、DEFAULT、MPEG_4、RAW_AMR 等,详情请查看 MediaRecorder.OutputFormat 类的 API 文档。Android 支持的音频编码格式有 DEFAULT、AMR_NB、AMR_WB、AAC 等。详情请查看 MediaRecorder.AudioEncoder 类的 API 文档。

Android 2.2 及其以后版本采用下面的方式设置输出格式和编码格式:

```
myRecorder.setProfile(CamcorderProfile.get(CamcorderProfile.QUALITY_HIGH));
```

步骤 5: 调用 MediaRecorder.setOutputFile() 设置 mediaRecorder 的输出文件名称。例如:

```
File videoFile = new File(Environment.getExternalStorageDirectory(), System.  
    currentTimeMillis() + ".3gp");  
myRecorder.setOutputFile(videoFile.getAbsolutePath());
```

在配置 MediaRecorder 的过程中,需要注意参数设置的顺序,否则应用程序可能会抛出 java.lang.IllegalStateException 异常。

步骤 6: 配置完成以后,调用 MediaRecorder.prepare() 准备录制。这个方法执行完后 MediaRecorder 就准备好了捕捉和编码音视频数据。

调用 MediaRecorder.start() 方法开始录制。录制完成后,可以调用 MediaRecorder.stop() 方法停止录制。当 MediaRecorder 对象完成音视频录制,并且不再使用时,调用 release() 方法对 MediaRecorder 资源进行释放。

【例 10 6】 工程 Demo_10_MediaRecorderForAudio 演示了利用 MediaRecorder 对象录制音频,并将录制的音频存储成文件。

录制音频需要获得录制 audio 的权限,保存录音文件需要向外部存储写数据的权限,回放录音文件需要从外部存储读数据的权限。在 AndroidManifest.xml 配置文件中添加权限声明,如代码段 10-9 所示。

代码段 10-9 声明权限

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>  
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>  
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

定义 MainActivity 类,界面如图 10 7 所示。界面中设置了 4 个按钮:点击“开始”按钮,开始录音;点击“停止”按钮,录音结束,存储录音文件;点击“播放”按钮,则播放先前录制的文件;点击“结束”按钮,则关闭 Activity,返回录制的音频的 URI。

MainActivity 的实现如代码段 10-10 所示。



图 10-7 录制音频文件

代码段 10-10 利用 MediaRecorder 对象录制音频

//package 和 import 语句略

```

public class MainActivity extends AppCompatActivity implements View.
    OnClickListener {
    private TextView stateView,saveView;
    private Button btnStart,btnStop,btnPlay,btnFinish;
    private MediaRecorder myRecorder;
    private MediaPlayer player;
    private File audioFile;
    private Uri fileUri;
    private boolean isRecord = false;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        stateView = (TextView)this.findViewById(R.id.view_state);
        saveView = (TextView)this.findViewById(R.id.view_save);
        stateView.setText("准备开始");
        btnStart = (Button)this.findViewById(R.id.btn_start);
        btnStop = (Button)this.findViewById(R.id.btn_stop);
        btnPlay = (Button)this.findViewById(R.id.btn_play);
        btnFinish = (Button)this.findViewById(R.id.btn_finish);
        btnStop.setEnabled(false);
        btnPlay.setEnabled(false);
        btnFinish.setEnabled(false);
        btnStart.setOnClickListener(this);
        btnStop.setOnClickListener(this);
        btnFinish.setOnClickListener(this);
        btnPlay.setOnClickListener(this);
    }
    public void onClick(View v) {
        int id = v.getId();

```

```
switch(id){
    case R.id.btn_start:
        //开始录制,先实例化一个 MediaRecorder 对象,然后进行相应的设置
        myRecorder = new MediaRecorder();
        //指定 AudioSource 为 MIC,从麦克风获取音频,这是最常用的
        myRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
        //指定输出格式和编码方式
        myRecorder.setOutputFormat(MediaRecorder.OutputFormat.
            DEFAULT);
        myRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.
            DEFAULT);
        //设置录音文件的存储位置
        audioFile = new File(Environment.getExternalStorageDirectory(),
            System.currentTimeMillis() + "aa.3gp");
        myRecorder.setOutputFile(audioFile.getAbsolutePath());
        try{
            myRecorder.prepare(); //缓冲
        } catch (IllegalStateException e1) {
            e1.printStackTrace();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        myRecorder.start(); //开始录制
        isRecord = true; //正在录制设为 true
        stateView.setText("正在录制");
        saveView.setText("录音文件保存在:" + audioFile.getAbsolutePath());
        btnStart.setEnabled(false);
        btnPlay.setEnabled(false);
        btnStop.setEnabled(true);
        break;
    case R.id.btn_stop:
        myRecorder.stop();
        myRecorder.release();
        //录制结束后,实例化一个 MediaPlayer 对象,然后准备播放
        player = new MediaPlayer();
        player.setOnCompletionListener(new MediaPlayer.
            OnCompletionListener() {
                @Override
                public void onCompletion(MediaPlayer arg0) { //更新按钮状态
                    stateView.setText("准备录制");
                    btnPlay.setEnabled(true);
                    btnStart.setEnabled(true);
                    btnStop.setEnabled(false);
                }
            });
    }
```

```
        }  
    });  
    try {  
        //准备播放  
        player.setDataSource(audioFile.getAbsolutePath());  
        player.prepare();  
    } catch (IllegalArgumentException e) {  
        e.printStackTrace();  
    } catch (IllegalStateException e) {  
        e.printStackTrace();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    stateView.setText("准备播放");  
    btnPlay.setEnabled(true);           //更新按钮状态  
    btnStart.setEnabled(true);  
    btnStop.setEnabled(false);  
    break;  
case R.id.btn_play:  
    //播放录音。录音结束时已经实例化 MediaPlayer,做好了播放的准备  
    player.start();  
    stateView.setText("正在播放");  
    btnStart.setEnabled(false);         //更新按钮状态  
    btnStop.setEnabled(false);  
    btnPlay.setEnabled(false);  
    break;  
case R.id.btn_finish:  
    //完成录制,返回录制的音频的 URI  
    Intent intent = new Intent();  
    intent.setData(fileUri);  
    this.setResult(RESULT_OK, intent);  
    this.finish();  
    break;  
    }  
}  
}
```

10.3 基于 Camera 类的图片摄取

10.3.1 Camera 类

Camera 是 Android 系统定义的摄像头类,它可以用于图像预览、捕获图片和录制视频等。在照相时,这个类也可以用来设置摄像头参数。

Camera 对象的常用方法如表 10-1 所示。

表 10-1 Camera 对象的常用方法

方 法 名	说 明
open()	获取 Camera 实例
getParameters()	获取 Camera 的参数
setParameters(param)	设置 Camera 的参数
setPreviewDisplay(holder)	Camera 与 SurfaceHolder 联系起来,设置预览窗口
release()	释放 Camera
startPreview()	启动预览功能
stopPreview()	停止预览

Camera 中含有一个内部类 Camera.Parameters,利用该类可以对 Camera 的参数进行设置。可以调用 getParameters()方法获得 Camera 的默认设置参数 Camera.Parameters,更改后用 setParameters(Camera.Parameters)方法对 Camera 重新进行设置。由于不同设备的 Camera 参数是不同的,所以在设置时,需要首先判断设备对应的参数,再加以设置。例如,在调用 setEffects()方法之前,最好先调用 getSupportedColorEffects()方法判断设备支持的参数,如果设备不支持颜色特性,那么该方法将返回一个 null。

10.3.2 利用 Camera 类实现图片的摄取

利用 Camera 类可以实现图片的摄取。拍照过程中的预览功能需要一个存放取景器的容器,这个容器就是 SurfaceView。使用 SurfaceView 的同时,还需要使用 SurfaceHolder。SurfaceHolder 相当于一个监听器,可以监听 Surface 上的变化,通过其内部类 Callback 来实现。

如果要在应用程序中使用 Camera,必须在 AndroidManifest.xml 配置文件中声明相应的 Camera 权限。如果用到了 Camera 和 auto focus 特征,还应该设置 android.hardware.camera 和 android.hardware.camera.autofocus 权限。格式如下:

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera"/>
<uses-feature android:name="android.hardware.camera.autofocus"/>
```

通常使用 SurfaceView 作为取景的容器和预览窗口,并通过调用 SurfaceView 的 getHolder()方法获得其控制器 SurfaceHolder。SurfaceHolder 是系统提供的控制 SurfaceView 的控制器,通过其内部类 SurfaceHolder.Callback 来实现监听变化。Camera 可以通过调用 setPreviewDisplay(SurfaceView)方法来设置 camera 的预览窗口。例如:

```
surfaceView= (SurfaceView) findViewById(R.id.myCameraView);
SurfaceHolder myholder = surfaceView.getHolder();
```

在得到了 SurfaceHolder 实例对象后,通过调用 SurfaceHolder 对象的 addCallback() 方法,实现将 SurfaceView 的回调接口 SurfaceHolder.Callback 绑定在 SurfaceHolder 上的功能。Callback 接口必须重写 3 个方法: SurfaceCreated()、SurfaceChanged() 和 SurfaceDestroyed(),如代码段 10-11 所示。这 3 个方法分别在 SurfaceView 被创建后、SurfaceView 发生变化时、SurfaceView 销毁时调用。

代码段 10-11 调用 addCallback()方法

```
mSurfaceHolder.addCallback(new Callback() {  
    public void surfaceDestroyed(SurfaceHolder holder) {  
        //具体代码略  
    }  
    public void surfaceCreated(SurfaceHolder holder) {  
        //具体代码略  
    }  
    public void surfaceChanged(SurfaceHolder holder, int format, int width,  
        int height) {  
        //具体代码略  
    }  
});
```

为了实现照片预览功能,需要将 SurfaceHolder 的类型设置为 PUSH,这可以通过调用 SurfaceHolder 的 setType() 方法来实现,例如:

```
myholder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

设置 Camera 的预览窗口完成后,就可以获得 Camera 实例进行预览了。具体方法是重写 SurfaceHolder.Callback 的 SurfaceCreated() 方法。当 SurfaceView 创建时,该方法被调用。通过 Camera 的静态方法 open() 可以获得 Camera 实例,然后设置 Camera 的预览窗口,最后通过调用 Camera 的 startPreview() 方法开始预览。具体实现方法如代码段 10-12 所示。

代码段 10-12 设置预览窗口

```
public void surfaceCreated(SurfaceHolder holder) {  
    myCamera = Camera.open(); //获取 Camera 实例  
    try {  
        myCamera.setPreviewDisplay(holder); //设置预览窗口  
    } catch (Exception e) {  
        e.printStackTrace();  
        myCamera.release(); //如果出现异常,则释放 Camera 对象  
    }  
    myCamera.startPreview(); //启动预览功能  
}
```

在拍照结束且不需要预览时,可以调用 `stopPreview()` 方法停止预览,同时也要调用 `release()` 方法将 `Camera` 实例销毁,这些一般在前面提到的 `SurfaceDestroyed()` 方法中实现,如代码段 10-13 所示,其中的 `ca` 是 `Camera` 实例对象的名称。

代码段 10-13 停止预览

```
public void surfaceDestroyed(SurfaceHolder holder) {  
    ca.stopPreview();  
    ca.release();  
    ca=null;  
}
```

调用 `Camera` 的 `takePicture()` 方法可以完成拍照,获取图片。`takePicture()` 方法中的参数 `shutter` 是 `Camera.ShutterCallback` 类型数据,是相机快门的回调方法,主要目的是通过声音提醒用户照片已经拍照完毕;参数 `jpeg` 是 `Camera.PictureCallback` 类型数据,是相机拍照数据的处理回调方法参数之一,用来将数据流转化为指定的图片格式并进一步处理。根据需要可将照片保存到媒体库、放到 `Activity` 中回显或做其他的处理。

当相机摄取照片时,依次执行方法的 4 个回调方法,在 `ShutterCallback` 中需要重写 `onShutter()` 方法,代码段 10-14 是一个示例。

代码段 10-14 摄取照片

```
Camera.ShutterCallback shutter = new ShutterCallback() { //实例化  
    public void onShutter() {  
        //相关处理逻辑  
    }  
};
```

同时还需要实现 `Camera.PictureCallback` 接口,重写 `onPictureTaken(byte[] data, Camera camera)` 方法处理获取的图片。代码段 10-15 是一个示例。

代码段 10-15 重写 `onPictureTaken()` 方法

```
Camera.PictureCallback jpeg = new PictureCallback() { //实例化  
    public void onPictureTaken(byte[] data, Camera camera) {  
        Bitmap bitmap = BitmapFactory.decodeByteArray(data, 0, data.length);  
        iv.setImageBitmap(bitmap); //iv 是 ImageView 的实例对象名  
        iv.setVisibility(View.VISIBLE); //使视图可见  
        //略  
    }  
};
```

代码中的 `BitmapFactory.decodeByteArray(data, 0, data.length)` 方法是照片数据流 `data` 转化为 `bitmap` 图片,并调用 `ImageView` 对象的 `setImageBitmap()` 方法将图片显示在手机屏幕上。

至此,相机的预览和摄取照片功能就可以实现了。当然,相机有不同的类型,也有不

同参数,还可对相机的参数进行设置,这时就需要更改 Camera.Parameters,可调用 getParameters() 方法获得 Camera 的默认参数 Camera.Parameters,更改后调用 setParameters(Camera.Parameters)方法对 Camera 重新进行设置。相关代码如代码段 10-16 所示。代码中的 setPictureFormat(PixelFormat.JPEG)的功能是设置图片的格式, params.set("rotation", 90)的功能是设置图片的旋转角度。另外还可以设置图片显示方式为横向、竖向或某个角度。可调用 setDisplayOrientation(int)方法设置照片与垂直显示边框的夹角。

代码段 10-16 参数设置

```
Parameters params = mCamera.getParameters();           //获得 Camera 的参数
params.setPictureFormat(PixelFormat.JPEG);              //设置图片格式
params.set("rotation", 90);                             //设置照片旋转 90°
mCamera.setParameters(params);                          //设置 Camera 的参数
int result = 90;
camera.setDisplayOrientation(result);                   //设置 camera 顺时针旋转的角度
```

【例 10 7】 工程 Demo_10_GetPhotoByCamera 演示了采用基于 Camera 类的方法实现拍照,主要实现了预览、点击预览图片时摄取照片并将照片存储在媒体库中的操作。

Camera 的生命周期和 SurfaceView 的生命周期保持一致, SurfaceView 创建时,创建 Camera 实例, SurfaceView 销毁时销毁 Camera 实例。该程序运行后,首先显示预览画面,点击预览画面就会获取照片并保存,如图 10 8 所示。由于涉及 Camera 硬件的支持,在真实设备上才能看到正确的效果。

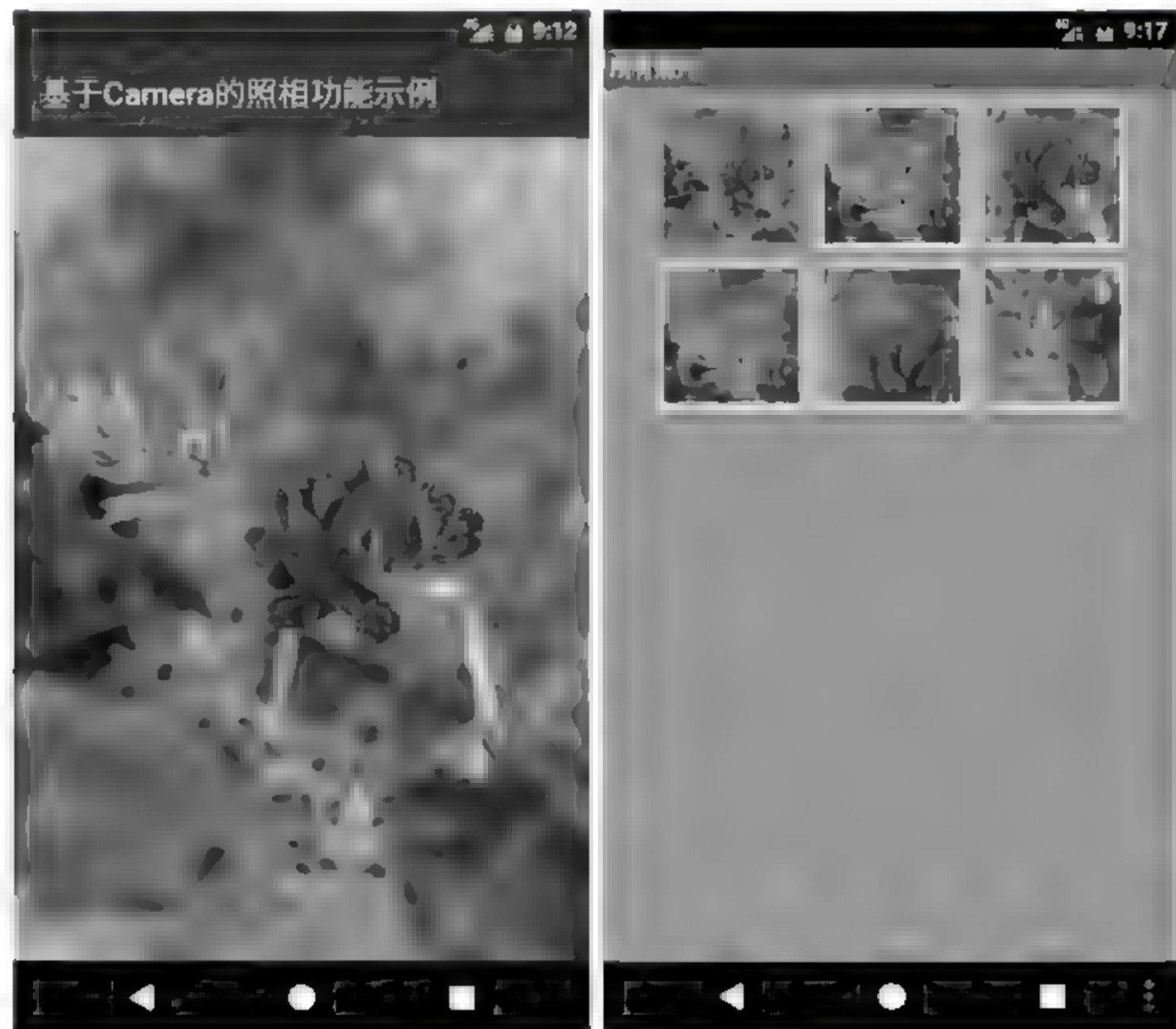


图 10-8 预览和拍摄照片

获取照片通过调用 Camera 的 `takePicture()` 方法实现,这需要实现 `Camera.PictureCallback` 接口并重写其 `onPictureTaken()` 方法。在该方法中实现获取照片的处理。本例中,在 `onPictureTaken()` 方法中实现了照片摄取后的保存功能,照片存储为 JPG 格式,存储在媒体库中。

MainActivity 类的主要代码如代码段 10-17 所示。

代码段 10-17 基于 Camera 类的方法实现拍照

//package 和 import 语句略

```
public class MainActivity extends AppCompatActivity {
    private SurfaceView surfaceView;
    private Camera myCamera=null; //android.hardware.Camera 对象
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this setContentView(R.layout.camera);
        this.setTitle("基于 Camera 的照相功能示例");
        surfaceView = (SurfaceView) findViewById(R.id.myCameraView);
        surfaceView.setFocusable(true);
        surfaceView.setFocusableInTouchMode(true);
        surfaceView.setClickable(true);
        surfaceView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Camera.ShutterCallback shutter = new ShutterCallback() {
                    //相机的快门回调接口
                    public void onShutter() {
                        //TODO Auto-generated method stub
                    }
                };
                PictureCallback jpeg = new PictureCallback() {
                    public void onPictureTaken(byte[] data, Camera camera) {
                        //data 是一个原始的 JPEG 图像数据
                        Uri imageUri = MainActivity.this.getContentResolver().
                            insert(MediaStore.
                                Images.Media.EXTERNAL_CONTENT_URI, new
                                    ContentValues());
                        try {
                            OutputStream os =
                                MainActivity.this.getContentResolver().
                                    openOutputStream(imageUri);
                            os.write(data);
                            os.flush();
                            os.close();
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }
                };
            }
        });
    }
}
```

```

        }
        myCamera.startPreview();
        //保存图片后,再次调用 startPreview()回到预览状态
    }
};
myCamera.takePicture(shutter, null, jpeg);
}
});
//SurfaceView中的 getHolder 方法可以获取一个 SurfaceHolder 实例
SurfaceHolder myholder = surfaceView.getHolder();
//为了实现照片预览功能,需要将 SurfaceHolder 的类型设置为 PUSH
myholder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
//设置回调函数, SurfaceHolder.Callback
myholder.addCallback(new Callback() {
    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int
        w, int h) {
    }
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        //当 Surface 被创建时,该方法被调用,可以在这里实例化 Camera 对象
        int i=Camera.getNumberOfCameras();
        myCamera = Camera.open(); //获取 Camera 实例
        try {
            Parameters pa =myCamera.getParameters();
            pa.setPictureFormat(ImageFormat.JPEG);
            pa.setPreviewSize(480,320);
            myCamera.setParameters(pa);
            myCamera.setPreviewDisplay(holder);
            myCamera.startPreview();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        //当 Surface 被销毁的时候,该方法被调用,在这里需要释放 Camera 资源
        myCamera.stopPreview();
        myCamera.release();
        myCamera=null;
    }
});
}
}

```

10.3.3 利用系统内置的 Camera 应用实现图片的摄取

通过调用 Android 系统内置的 Camera 应用也可以摄取图片。这时只需要指定一个 `MediaStore.ACTION_IMAGE_CAPTURE` 的 Action 来启动 Camera 应用即可。

【例 10-8】 工程 Demo_10 UseCamera 演示了通过调用系统内置的 Camera 应用实现摄取照片。程序实现了照片的摄取、保存并回放显示。MainActivity 的实现如代码段 10-18 所示。

代码段 10-18 调用系统内置的 Camera 应用实现摄取照片

//package 和 import 语句略

```
public class MainActivity extends AppCompatActivity implements View.  
    OnClickListener {  
    private ImageView imageView;  
    private Uri imageUri;  
    Button Btn1;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        this.setTitle("调用系统内置的 Camera 应用摄取图像示例");  
        imageView= (ImageView)this.findViewById(R.id.myImage);  
        Btn1= (Button)this.findViewById(R.id.btn_capture);  
        Btn1.setOnClickListener(this);  
    }  
    public void onActivityResult(int requestCode, int resultCode, Intent data){  
        if(resultCode == RESULT_OK){  
            Btn1.setText("继续拍摄");  
            Bundle extras =data.getExtras();  
            Bitmap bmp = (Bitmap) extras.get("data");//获取返回的图像  
            imageView.setImageBitmap(bmp);           //回显拍摄的照片  
        }  
    }  
    public void onClick(View v){                               //调用系统内置的照相机应用  
        int id =v.getId();  
        if(id == R.id.btn_capture){  
            Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);  
            startActivityForResult(intent, 1);  
        }  
    }  
}
```

点击主界面的按钮,会通过调用 `startActivityForResult()` 方法启动 Android 系统内置的 Camera 应用,屏幕显示摄像头预览的画面,如图 10-9(a)所示。拍摄完成后返回 MainActivity 界面,获取拍摄照片数据后将其显示在一个 `ImageView` 控件中。拍摄完一张照片后的程序界面如图 10-9(b)所示。



(a) 屏幕显示摄像头预览画面

(b) 拍摄完一张照片后的程序界面

图 10-9 调用系统内置的 Camera 应用摄取图片

10.4 本章小结

本章介绍了在 Android 系统如何处理和使用音视频、图片等资源。在处理和使用这些多媒体资源时,可以使用 `MediaPlayer` 对象、`MediaRecorder` 对象、`VideoView` 对象或 `Camera` 对象,也可以使用 Android 系统内置的播放器、录音或照相程序。学习本章内容要重点掌握音视频播放和录制的方法以及图片的摄取方法,并能够编写简单的多媒体应用程序。

习 题

1. 设计一个用于注册的 Activity。要求界面中的注册项包括用户名、密码、照片,界面中有“拍照”和“注册”两个按钮,点击“拍照”按钮,开始拍摄照片,并将照片存储为外部文件,同时回显到界面中。当用户点击“注册”按钮后,将用户名、密码和照片的 URI 路径

存储到 SharedPreferences。

2. 设计一个音乐播放器,能播放、暂停、停止音乐,播放过程中能显示音乐文件的名称,能选择上一首/下一首音乐播放。

3. 设计一个音乐播放器,能显示媒体库全部音乐的列表,点击列表中的某个文件即开始播放,播放过程中能显示音乐文件的名称。

第 11 章

Web 应用开发

Android 提供了多种方式来利用 Internet 资源。可以使用客户端 API 直接与服务器远程交互,常用的方法有利用 URLConnection、HttpURLConnection 或 Socket 与远程服务器交互;也可以使用 WebView 控件在 Activity 中包含一个基于 WebKit 的浏览器,利用浏览器访问网络资源。本章主要介绍这些访问 Internet 资源的方法。

11.1 Android 网络通信概述

Android 基于 Linux 内核,它包含一组网络通信功能,提供了多个类来帮助处理网络通信。目前,Android 平台主要有 3 种网络接口可以使用,它们分别是 java.net.* (标准 Java 接口)、org.apache.* (Apache 接口)和 android.net.* (Android 网络接口)。Android SDK 中提供的与网络有关的包如表 11-1 所示。

表 11-1 Android SDK 中与网络有关的包

包	功能描述
java.net.*	提供与网络通信相关的类,包括流和数据包 Socket、Internet 协议和常见 HTTP 处理
java.io	虽然没有提供现实网络通信功能,但该包中的类由其他 Java 包中提供的 socket 和链接使用。它们还用于与本地文件的交互
java.nio	包含表示特定数据类型的缓冲区的类。适用于两个基于 Java 语言的端点之间的通信
org.apache.*	表示许多为 HTTP 通信提供精确控制和功能的包。可以将 Apache 视为开源 Web 服务器
android.net.*	除核心 java.net.* 类以外,包含额外的网络访问 Socket。该包包括 URI 类,后者经常用于 Android 应用程序,而不仅仅是传统的网络操作
android.net.http	包含处理 SSL 证书的类

1. 标准 Java 接口

Java.net.* 提供与联网有关的类和接口,包括流和数据包套接字、Internet 协议、常见 HTTP 协议处理。这些类和接口提供了访问 HTTP 服务的基本功能,包括创建 URL 对象和 URLConnection 对象、设置连接参数、连接到服务器、向服务器写入数据以及从服

务器读取数据等。其通信可以采用 GET 和 POST 两种方式来实现。URLConnection 对象表示应用程序和 URL 之间的通信连接。程序可以通过它的实例向该 URL 发送请求, 读取 URL 引用的资源。

例如代码段 11-1 创建了 URL 对象和 HttpURLConnection 对象, HttpURLConnection 是 URLConnection 的子类。代码中设置了连接参数, 连接到服务器并从服务器读取了数据。

代码段 11-1 从服务器读取数据

```
try {
    URL url = new URL("http://www.baidu.com/");           //创建 URL 对象
    HttpURLConnection myconnection = (HttpURLConnection)url.openConnection();
    //创建 URL 连接
    myconnection.setConnectTimeout(10000);                 //设置参数
    myconnection.connect();                                 //连接服务器
    InputStream is = myconnection.getInputStream();          //取得数据
    ...                                                      //处理数据
} catch (IOException e) {
    e.printStackTrace();
}
```

2. Apache 接口

HttpClient 是 Apache Jakarta Common 下的子项目, 它是一个开源项目, 弥补了 java.net.* 灵活性不足的缺点, 为客户端的 HTTP 编程提供高效、功能丰富的工具包支持, 并且它支持 HTTP 协议最新的版本和建议。Android 平台引入了 ApacheHttpClient 的同时还提供了对它的一些封装和扩展, 例如设置默认的 HTTP 超时和缓存大小等。Android 平台用的版本是 HttpClient 4.0。对于 HttpClient 类, 可以使用 HttpPost 和HttpGet 类以及 HttpResponse 来进行网络连接。

使用这部分接口的操作方法与 java.net.* 基本类似, 主要包括创建 HttpClient、GetMethod /PostMethod 以及 HttpResponse 等对象、设置连接参数、执行 HTTP 操作、处理服务器返回结果等。

需要注意的是, 在 Android 6.0 (API 23) 中已经移除了 Apache HttpClient 相关的类, 而推荐使用 HttpURLConnection, 如果要继续使用这些类, 需要导入相关的包。

3. Android 网络接口

Android.net.* 包实际上是通过在 Apache 中 HttpClient 的封装来实现的一个 HTTP 编程接口, 同时还提供了 HTTP 请求队列管理以及 HTTP 连接池管理, 以提高并发请求情况下的处理效率, 除此之外还有网络状态监视等接口、网络访问的 Socket、常用的 URI 类以及 WiFi 相关的类等。

代码段 11-2 是一个通过 AndroidHttpClient 访问服务器的示例。

代码段 11-2 通过 AndroidHttpClient 访问服务器

```
try {
    AndroidHttpClient client=AndroidHttpClient.newInstance("my_agent");
    HttpGet httpGet =new HttpGet ("http://www.test.com/");
    //创建 HttpGet 对象,该对象会自动处理 URL 地址的重定向
    HttpResponse response =client.execute(httpGet);
    if (response.getStatusLine().getStatusCode() ==HttpStatus.SC_OK) {
        ...                //处理数据
    }
    else{
        ...                //错误处理
    }
    client.close();        //关闭连接
} catch (Exception e) {
    ...                //异常处理
}
```

4. 使用 WebView 控件访问网络

在 Android 中,访问网页数据有两种形式。一种是使用移动设备上的浏览器直接访问的网络应用程序,这种情况用户不需要额外安装其他应用,只要有浏览器就行;另一种是在用户的移动设备上安装客户端应用程序(.apk),并在此客户端程序中嵌入 WebView 控件来显示从服务器端下载的网页数据。对于前者来说,主要的工作是根据移动设备客户端的屏幕来调整网页的显示尺寸、比例等;而后者需要单独开发基于 WebView 的 Web 应用程序。WebView 控件的详细使用方法见 11.3 节。

11.2 网络资源的访问

由于需要访问网络,本章的操作都需要在 AndroidManifest 配置文件中声明访问网络的权限:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

另外需要注意的是,Android 4.0 之后系统强制性地不允许在主线程访问网络,否则会出现 android.os.NetworkOnMainThreadException 异常,所以应该在子线程中访问网络。

11.2.1 使用 HTTP 的 GET 方式访问网络

HTTP 是 Web 联网的基础,也是移动设备联网常用的协议之一。HTTP 协议是建立在 TCP 协议之上的一种协议,主要用于 Web 浏览器和 Web 服务器之间的数据交换。

HTTP 连接最显著的特点是客户端发送的每次请求都需要服务器回送响应,在请求结束后,会主动释放连接。客户向服务器请求服务时,只需传送请求方法和路径,常用的请求方法有 GET、POST、HEAD 等。有关 HTTP 的详细介绍,读者可以查阅 RFC 2616 或 <http://www.chinaw3c.org/>。

URL 类位于 java.net 包下,使用的资源可以是简单的文件或目录,也可以是对更复杂的对象的引用。URL 由协议名、主机、端口和资源路径组成。

URLConnection 是抽象类,无法直接实例化对象。其对象主要通过 URL 的 openConnection() 方法获得。通常的操作方式是,先通过 URL 对象的 openConnection() 方法获取一个 URLConnection 对象,然后调用其 getInputStream() 方法打开一个 Internet 数据流,读入数据。

【例 11-1】 工程 Demo_11_HttpGetConnection 演示了如何使用 HTTP 的 GET 方式从网络中获取数据。

本例中使用有道翻译 API,在访问网站之前需要申请一个 API key,如图 11-1 所示,申请网址为 <http://fanyi.youdao.com/openapi?path=data=mode>。



图 11-1 申请 API key

MainActivity.java 的主要代码如代码段 11-3 所示,运行结果如图 11-2 所示。

代码段 11-3 使用 HTTP 的 GET 方式从网络中获取数据

//package 和 import 语句略

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {
```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
String urlString= "http://fanyi.youdao.com/openapi.do?keyfrom=
    DemoHttpURLTest&key=846100214&type=data&doctype=xml&version=1.1&q=
    good";
new AsyncTask<String,Void,Void> () {
    @Override
    protected Void doInBackground(String... params) {
        try {
            URL myUrl=new URL(params[0]);
            URLConnection conn =myUrl.openConnection();
                                                    //获取 URLConnection 对象
            InputStream inputStream = conn.getInputStream();
                                                    //读数据,得到的是字节流
            InputStreamReader inputStreamReader =new InputStreamReader
                (inputStream, "UTF-8");           //包装为字符流
            BufferedReader bufferedReader=new BufferedReader
                (inputStreamReader);
            String strLine="";
            while ((strLine=bufferedReader.readLine())!=null){
                System.out.println("读取到 -- "+ strLine);
            }
            bufferedReader.close();
            inputStreamReader.close();
            inputStream.close();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}.execute(urlString);
}
}

```

11.22 使用 HTTP 的 POST 方式访问网络

【例 11 2】 工程 Demo_11_HttpPostConnection 演示了使用 HTTP 的 POST 方式从网络中获取数据,实现例 11-1 的功能。

MainActivity.java 的主要代码如代码段 11-4 所示,运行结果与例 11-1 相同。



图 11-2 使用 HTTP 的 GET 访问网络数据

代码段 11-4 采用 Post 方式从网络中获取数据

//package 和 import 语句略

```

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        String urlString="http://fanyi.youdao.com/openapi.do";
        new AsyncTask<String,Void,Void> () {
            @Override
            protected Void doInBackground(String... params) {
                try {
                    URL myUrl=new URL(params[0]);
                    HttpURLConnection conn = (HttpURLConnection)myUrl.
                        openConnection();
                    conn.setDoOutput(true);
                    conn.setRequestMethod("POST"); //设置为 POST 方式
                    OutputStreamWriter outputStreamWriter=
                        new OutputStreamWriter(conn.getOutputStream());
                    BufferedWriter bufferedWriter=new BufferedWriter
                        (outputStreamWriter);
                    bufferedWriter.write("keyfrom= DemoHttpURLTest&key=
                        846100214&type= data&doctype= xml&version= 1.1&q= good");
                    bufferedWriter.flush();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }.execute(urlString);
    }
}

```

```

        InputStream inputStream = conn.getInputStream();
                                                //读数据,得到的是字节流
        InputStreamReader inputStreamReader = new InputStreamReader
            (inputStream, "UTF-8");          //包装为字符流
        BufferedReader bufferedReader = new BufferedReader
            (inputStreamReader);
        String strLine = "";
        while ((strLine = bufferedReader.readLine()) != null) {
            System.out.println("读取到 -- " + strLine);
        }
        bufferedReader.close();
        inputStreamReader.close();
        inputStream.close();
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
}.execute(urlString);
}
}

```

11.2.3 使用 HttpURLConnection 访问网络

HttpURLConnection 是 URLConnection 的子类,二者都位于 java.net 包内。与 URLConnection 类似,HttpURLConnection 也是抽象类,无法直接实例化对象,主要通过 URL 的 openConnection() 方法获得。

通常 HttpURLConnection 的实现步骤如下。

步骤 1: 通过调用 URL.openConnection() 方法得到 HttpURLConnection 对象,设置请求头属性,如数据类型、数据长度等。如果采用 POST 方式传送数据,则需要设置 setDoOutput(true),该属性值默认为 false。

步骤 2: 浏览器向服务器发送数据,例如提交 form 表单或者向服务器发送一个文件。

步骤 3: 读取服务器发来的响应,包括 servlet 写进 response 的头数据(content type 及 content-length 等)和 body 数据等。

步骤 4: 调用 HttpURLConnection 的 disconnect() 方法,释放资源。

使用 HttpURLConnection 对象可以实现 HTTP 连接,具体的用途包括获取 HTML 源码、获取网络图片、获取 XML、发送 GET 请求或 POST 请求、上传文件等。例如,代码段 11-5 实现了采用 POST 方式发送 XML 数据。XML 格式是通信的标准语言,Android 系统可以通过发送 XML 文件传输数据。发送 POST 请求必须设置允许输出和一系列

Request 参数,最好不要使用缓存。

代码段 11-5 采用 POST 方式发送 XML 数据

```
byte[] xmlbyte = xml.toString().getBytes("UTF-8");    //将 XML 文件写入到 byte 数组中
URL url = new URL("http://10.0.2.2:8080/test/contanct.do?method=readxml");
                                                    //创建 URL 对象,并指定地址和参数
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setDoOutput(true);                            //设置允许输出
conn.setUseCaches(false);                          //设置不使用缓存
conn.setRequestMethod("POST");                      //设置以 POST 方式传输
conn.setRequestProperty("Connection", "Keep-Alive");
                                                    //维持长连接
conn.setRequestProperty("Charset", "UTF-8");        //设置字符集
conn.setRequestProperty("Content-Length", String.valueOf(xmlbyte.length));
                                                    //设置文件的总长度
conn.setRequestProperty("Content-Type", "text/xml; charset=UTF-8");
                                                    //设置文件类型
OutputStream outStream = conn.getOutputStream();
outStream.write(xmlbyte);                          //以文件流的方式发送 XML 数据
```

【例 11 3】 工程 Demo_11_HttpURLConnection 演示了如何使用 HttpURLConnection 的 GET 方式从网络中获取数据。

本例完成与例 11-1 相同的功能, MainActivity.java 的主要代码如代码段 11-6 所示。

代码段 11-6 使用 HttpURLConnection 的 GET 方式从网络中获取数据

```
//package 和 import 语句略
public class MainActivity extends AppCompatActivity {
    HttpURLConnection conn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        String urlString="http://fanyi.youdao.com/openapi.do?keyfrom=
            DemoHttpURLTest&key= 846100214&type= data&doctype= xml&version= 1.1&q=
            good";
        new AsyncTask<String,Void,Void> () {
            @Override
            protected Void doInBackground(String... params) {
                try {
                    URL myUrl=new URL (params[0]);
                    //获取 HttpURLConnection 对象
                    conn = (HttpURLConnection)myUrl.openConnection();
                    conn.setRequestMethod("GET");
```

```

        conn.setConnectTimeout(5000);
        conn.setUseCaches(false);           //不使用缓存
        if (conn.getResponseCode() == 200) {
            InputStream inputStream = conn.getInputStream();
                                                    //读数据,得到的是字节流
            InputStreamReader inputStreamReader = new InputStreamReader
                (inputStream, "UTF-8"); //包装为字符流
            BufferedReader bufferedReader = new BufferedReader
                (inputStreamReader);
            String strLine = "";
            while ((strLine = bufferedReader.readLine()) != null) {
                System.out.println("读取到 -- " + strLine);
            }
            bufferedReader.close();
            inputStreamReader.close();
            inputStream.close();
        }
        else {
            System.out.println("请求失败!");
        }

    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (conn != null) {
            //关闭连接即设置 http.keepAlive=false;
            conn.disconnect();
        }
    }
    return null;
}
}.execute(urlString);
}
}

```

【例 11-4】 工程 Demo_11_HttpURLConnection 演示了如何使用 HttpURLConnection 的 POST 方式从网络中获取数据。

本例完成与例 11-1 相同的功能, MainActivity.java 的主要代码如代码段 11-7 所示。

代码段 11-7 使用 HttpURLConnection 的 POST 方式从网络中获取数据

//package 和 import 语句略

```
public class MainActivity extends AppCompatActivity {
    HttpURLConnection conn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        String urlString="http://fanyi.youdao.com/openapi.do";
        new AsyncTask<String,Void,Void> () {
            @Override
            protected Void doInBackground(String... params) {
                try {
                    URL myUrl=new URL(params[0]);
                    conn = (HttpURLConnection)myUrl.openConnection();
                    conn.setDoOutput(true);
                    conn.setRequestMethod("POST"); //设置为 POST 方式
                    OutputStreamWriter outputStreamWriter=
                        new OutputStreamWriter(conn.getOutputStream());
                    BufferedWriter bufferedWriter=new BufferedWriter
                        (outputStreamWriter);
                    bufferedWriter.write("keyfrom= DemoHttpURLTest&key=
                        846100214&type=data&doctype=xml&version=1.1&q=good");
                    bufferedWriter.flush();
                    InputStream inputStream = conn.getInputStream();
                                                                    //读数据,得到的是字节流
                    InputStreamReader inputStreamReader = new InputStreamReader
                        (inputStream, "UTF-8"); //包装为字符流
                    BufferedReader bufferedReader=new BufferedReader
                        (inputStreamReader);
                    String strLine="";
                    while ((strLine=bufferedReader.readLine())!=null){
                        System.out.println("读取到 -- "+strLine);
                    }
                    bufferedReader.close();
                    inputStreamReader.close();
                    inputStream.close();
                } catch (MalformedURLException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                } finally {
                    conn.disconnect();
                }
                return null;
            }
        }
```

```
    }  
    }.execute(urlString);  
}  
}
```

在 Android 中对文件流进行操作时要注意,当文件较大时,最好将文件写到外部存储而不是直接写到手机内存上,因为手机内存的空间非常有限。另外,对文件流操作结束后要及时将其关闭。

在程序中可以设置连接超时,如果网络状态欠佳,超过默认时间,Android 系统会收回资源,中断操作,避免了程序长时间等待。另外,网络读写操作容易产生一些异常,所以在编写网络应用程序时最好捕捉每一个异常并采取相应措施。

对于每一次 HttpURLConnection 连接的状态,可以调用 HttpURLConnection.getResponseCode 方法取得当前网络连接的服务器应答代码,或调用 HttpURLConnection.getResponseMessage 取得返回的信息。常见的应答代码及其对应的信息如表 11-2 所示。

表 11-2 服务器应答代码

ResponseCode	ResponseMessage	说 明
200	OK	成功
401	Unauthorized	未授权
500	Internal Server Error	服务器内部错误
404	Not Found	找不到该网页

如果要获取网络图片,从 HttpURLConnection 对象中获取输入流读取数据后,调用 BitmapFactory 的 decodeByteArray(byte[] data, int offset, int length) 方法将数据转换为图片对象,就可以在 Activity 中使用了。如果获取的是 XML 数据,还需要使用 XmlPullParser 对其解析。出于篇幅原因,在此不再赘述,有兴趣的读者可以查阅相关文献。

11.24 使用 Socket 进行网络通信

Socket 是网络通信的一种接口。基于不同的协议,有各种不同的 Socket,如基于 TCP 协议的 Socket、基于 UDP 协议的 Socket、基于蓝牙协议的 Socket 等。Android 中使用的是 Java 的 Socket 模型,Socket 类在 java.net 包中。

使用 HttpURLConnection 发送数据时,由于系统内部的缓存机制,如果上传较大的文件,会导致内存溢出。这时可以使用 Socket 发送 TCP 请求,将上传数据分段发送。另一个重要的区别是,HTTP 连接使用的是“请求/响应”的方式,不仅在请求时需要先建立连接,而且需要客户端先向服务器发出请求,服务器端才能回复数据;而 Socket 在双方建立起连接后就可以直接进行数据的传输。

应用程序可以通过 Socket 向网络发送请求或者应答网络的请求,Socket 由两部分组

成,一部分是服务器端的 `ServerSocket`,这个 `Socket` 主要用来接收来自网络请求,它一直监听在某一个端口上。端口号取值的范围是 $0 \sim 65535$,自定义的应用程序通常使用 1124 以上的端口,以避免和其他应用程序的端口冲突。另一部分是客户端的 `ClientSocket`,这个 `Socket` 主要用来向网络发送数据。

通常在服务器端建立一个 `ServerSocket` 类的对象并绑定到一个端口上。`ServerSocket` 对象用于监听来自客户端的 `Socket` 连接,如果没有连接,它将一直处于等待状态。

`ServerSocket` 类常用的构造方法如下:

- `ServerSocket(int port)`: 用指定的端口 `port` 来创建一个 `ServerSocket`,`port` 参数必须是一个有效的端口整数值。使用该构造方法创建的 `ServerSocket` 没有指定 IP 地址,该 `ServerSocket` 将会绑定到本机默认的 IP 地址。
- `ServerSocket(int port, int backlog)`: 用指定的端口 `port` 和指定连接队列长度创建一个 `ServerSocket`。
- `ServerSocket(int port, int backlog, InetAddress addr)`: 在机器存在多个 IP 地址的情况下,允许通过 `addr` 这个参数来指定将 `ServerSocket` 绑定到哪一个 IP 地址。

另外要注意,由于手机无线上网的 IP 地址通常是由移动运营公司动态分配的,一般不会有自己固定的 IP 地址,因此很少在手机上运行服务器端,服务器端通常运行在有固定 IP 的服务器上。

建立了 `ServerSocket` 对象后,调用 `accept()` 方法进入阻塞监听状态,直到连接建立。如果接收到一个客户端 `Socket` 的连接请求,`accept()` 方法将返回一个与客户端连接 `Socket` 对应的 `Socket` 对象,然后创建一个线程给该 `Socket` 对象运行。否则该方法将一直处于等待状态,线程也被阻塞。通常情况下,服务器不应该只接收一个客户端请求,而应该不断地接收来自客户端的所有请求。

`ServerSocket` 对象接收连接请求后,双方就可以进行通信。通信完成后,`ServerSocket` 对象回到监听状态,继续监听客户端的请求。当 `ServerSocket` 使用完毕后,调用 `ServerSocket` 的 `close()` 方法来关闭该 `ServerSocket`。

而在客户端,首先创建客户端 `Socket`,指定服务器端 IP 地址与端口号。客户端通常使用 `Socket` 的构造方法来连接到指定服务器,`Socket` 类常用的构造方法如下:

- `public Socket(InetAddress address, int port)`: 用服务器端的 IP 地址对象和端口号建立 `Socket`。
- `public Socket(String host, int port)`: 用服务器端的机器名和端口号建立 `Socket`。

例如创建连接到本机服务器、9090 端口的 `Socket`:

```
Socket socket = new Socket("10.0.2.2", 9090);
```

当创建了客户端 `Socket` 后,就会连接到指定服务器,让服务器上的 `ServerSocket` 的 `accept()` 方法执行,于是服务器端和客户端就产生一对互相连接的 `Socket`。这样,Server

和 Client 就可以使用 Socket 进行通信了。

当服务器和客户端产生了对应的 Socket 之后,就可以通过 Socket 进行通信。Socket 提供两个方法来获取输入流和输出流,分别是 `getInputStream()` 和 `getOutputStream()`。`getInputStream()` 方法返回该 Socket 对象对应的输入流,让程序通过该输入流从 Socket 中取出数据;`getOutputStream()` 方法返回该 Socket 对象对应的输出流,让程序通过该输出流向 Socket 中输出数据。

当 Socket 使用完毕后,使用 `close()` 方法来关闭该 Socket。

【例 11-5】 工程 Demo_11_SocketToServer 演示了如何使用 Socket 进行网络通信。MainActivity 类实现客户端的 Socket,主要代码如代码段 11-8 所示。

代码段 11-8 实现客户端的 Socket

```
//package 和 import 语句略
public class MainActivity extends AppCompatActivity {
    EditText show;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if (android.os.Build.VERSION.SDK_INT > 9) {
            StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.
                Builder().permitAll().build();
            StrictMode.setThreadPolicy(policy);
        }
        show = (EditText) findViewById(R.id.show);
        try{
            Socket socket = new Socket("10.0.2.2", 9090);
            //创建连接到本机的服务器、9090 端口的 Socket
            BufferedReader br = new BufferedReader(new InputStreamReader
                (socket.getInputStream()));
            //将 Socket 对应的输入流包装成 BufferedReader
            String line = br.readLine(); //BufferedReader 数据转换为字符串
            show.setText("来自服务器的数据:\n" + line);
            br.close();
            socket.close();
            //关闭 Socket
        }catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

服务器端 Socket 的主要代码如代码段 11-9 所示。

代码段 11-9 实现服务器端 Socket

```
//package 和 import 语句略
public class MySocketServer {
    public static void main(String[] args) throws IOException {
        System.out.println("服务器启动...");
        ServerSocket ss = new ServerSocket(9090);
        //创建一个 ServerSocket,在 9090 端口监听客户端 Socket 的连接请求
        while (true){    //采用循环不断接收来自客户端的请求
            Socket s = ss.accept();
            //每当接收到客户端 Socket 的请求时,服务器端也对应产生一个 Socket
            OutputStream os = s.getOutputStream();
            //获取输出流,发送字符串
            os.write("Hello, Welcome!\n".getBytes("utf-8"));
            os.close(); //关闭输出流
            s.close();
            //关闭 Socket
        }
    }
}
```

运行 MySocketServer,此时服务器端 Socket 处于监听状态,直到接收到一个客户端 Socket 的连接请求。当与客户端创建了 Socket 连接后,服务器端 Socket 就会发送字符串“Hello, Welcome!\n”。而客户端的 Activity 运行后,客户端 Socket 会向服务器发出请求,建立连接后获取服务器端发出的数据,其运行结果如图 11-3 所示。



图 11-3 客户端 Socket 获取的数据

11.3 WebView

Android.webkit.WebView 继承自 Android.widget.AbsoluteLayout 类,用于加载和显示 Web 网页。WebView 控件可以被嵌入到应用程序中,实现一个基于 WebKit 浏览器的功能。

11.3.1 WebView 的基本用法

首先在布局文件中声明 WebView,如代码段 11-10 所示,然后在 Activity 中获取该 WebView 实例。也可以在 Activity 中直接使用 new 操作符实例化一个 WebView 对象。

代码段 11-10 在布局文件中声明 WebView

```
<WebView
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

取得 WebView 实例后,就可以调用 WebView 对象的 loadUrl()方法加载网页,如代码段 11-11 所示。

代码段 11-11 加载网页

```
mywebview = (WebView)findViewById(R.id.webview);
//获取 WebView 控件实例
mywebview.loadUrl("http://m.baidu.com/");
//加载需要显示的网页
```

如果 WebView 中需要用户手动输入用户名、密码或其他,则必须设置支持获取手势焦点:

```
webview.requestFocusFromTouch();
```

11.3.2 WebView 的参数设置

1. WebSettings

调用 WebView 对象的 getSettings()方法可以取得一个 WebSettings 对象,该对象用于设置 WebView 属性。WebSettings 的常用方法及其功能如表 11 3 所示。

表 11-3 WebSettings 的常用方法

方 法 名	功 能 说 明
setJavaScriptEnabled(true);	支持 JavaScript,能够执行 JavaScript 脚本
setPluginsEnabled(true);	支持插件
setUseWideViewPort(true);	将图片调整到适合 WebView 的大小
setLoadWithOverviewMode(true);	缩放至屏幕的大小,这样,在打开页面时可以自适应屏幕
setSupportZoom(true);	支持缩放,默认为 true
setBuiltInZoomControls(true);	设置内置的缩放控件。setSupportZoom(true)时该设置才有效

续表

方 法 名	功 能 说 明
<code>setDisplayZoomControls(false);</code>	隐藏原生的缩放控件
<code>setLayoutAlgorithm(LayoutAlgorithm.SINGLE_COLUMN);</code>	支持内容重新布局
<code>supportMultipleWindows();</code>	多窗口
<code>setCacheMode(WebSettings.LOAD_CACHE_ELSE_NETWORK);</code>	关闭 WebView 中的缓存
<code>setAllowFileAccess(true);</code>	设置可以访问文件
<code>setLoadsImagesAutomatically(true);</code>	支持自动加载图片
<code>setDefaultTextEncodingName("utf-8");</code>	设置编码格式
<code>setPluginState(PluginState.OFF);</code>	设置是否支持 Flash 插件
<code>setDefaultFontSize(20);</code>	设置默认字体大小

如果需要在 WebView 中使用 JavaScript,则需要设置 WebView 属性使其能够支持 JavaScript。然后,将 JavaScript 与 Android 客户端代码进行绑定,这样就可以由 JavaScript 调用 Android 代码中的方法。例如,JavaScript 代码想利用 Android 的代码来显示一个 Dialog,而不用 JavaScript 的 `alert()` 方法,这时就需要在 Android 代码和 JavaScript 代码间创建接口,从而可以在 Android 代码中实现显示对话框的方法,然后 JavaScript 调用此方法。绑定的具体方法如下。

创建 Android 代码和 JavaScript 代码的接口,即创建一个类,类中的方法将被 JavaScript 调用,如代码段 11-12 所示。

代码段 11-12 JavaScript 代码的接口

```
public class JavaScriptInterface {
    Context mContext;

    JavaScriptInterface(Context c) {
        //初始化 context,供 makeText 方法中的参数来使用
        mContext = c;
    }

    public void showToast(String toast) {
        //创建一个方法,实现显示对话框的功能,供 JavaScript 中的代码调用
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}
```

通过调用 `addJavascriptInterface()` 方法,把前面创建的接口类与运行在 WebView 上的 JavaScript 进行绑定。其中第二个参数是这个接口对象的名字,以方便 JavaScript 调用。

```
myWebView.addJavascriptInterface(new JavaScriptInterface(this), "Andr_Toast");
```

在 HTML 中的 JavaScript 部分调用 showToast()方法,如代码段 11-13 所示。

代码段 11-13 在 HTML 中的 JavaScript 部分调用 showToast()方法

```
<script type="text/javascript">
function showAndroidToast(toast) {
    Andr_Toast.showToast(toast);
}
</script>
<input type="button" value="hello" onClick="showAndroidToast('Hello
    Android!')"/>
```

2. WebViewClient

WebViewClient 是一个专门辅助 WebView 处理各种通知、请求等事件的类。通过继承 WebViewClient 并重载它的方法可以实现不同功能的定制。常用方法及其功能如表 11-4 所示。

表 11-4 WebViewClient 的常用方法

方 法 名	功 能 说 明
shouldOverrideUrlLoading(WebView view, String url)	在网页上的所有加载都经过这个方法,这个方法中可以做很多操作,如获取 URL,查看 url.contains("add"),进行添加操作
shouldOverrideKeyEvent (WebView view, KeyEvent event)	处理在浏览器中的按键事件
onPageStarted(WebView view, String url, Bitmap favicon)	开始载入页面时调用的,可以设定一个 loading 的页面,告诉用户程序在等待网络响应
onPageFinished(WebView view, String url)	在页面加载结束时调用,此时可以关闭 loading 进度条,切换程序动作等
onLoadResource(WebView view, String url)	在加载页面资源时会调用,每一个资源的加载都会调用一次
onReceivedError(WebView view, int errorCode, String description, String failingUrl)	报告错误信息
doUpdateVisitedHistory(WebView view, String url, boolean isReload)	更新历史记录
onFormResubmission(WebView view, Message dontResend, Message resend)	应用程序重新请求网页数据
onReceivedHttpAuthRequest(WebView view, HttpAuthHandler handler, String host,String realm)	获取返回信息授权请求

续表

方 法 名	功 能 说 明
onReceivedSslError(WebView view, SslErrorHandler handler, SslError error)	让 WebView 对象处理 HTTPS 请求
onScaleChanged(WebView view, float oldScale, float newScale)	WebView 对象发生改变时调用
onUnhandledKeyEvent(WebView view, KeyEvent event)	Key 事件未被加载时调用

如果需要在 WebView 中显示网页,而不是在内置浏览器中浏览,则需要调用 WebView 对象的 setWebViewClient() 方法设置 WebView,该方法要求传入一个 WebViewClient 对象。这个 WebViewClient 对象需要继承 WebViewClient 并重载 shouldOverrideUrlLoading() 方法。

3. WebChromeClient

Android 中还提供了一个类 WebChromeClient,专门用来辅助 WebView 处理 JavaScript 的对话框、网站图标、网站标题、加载进度等。同样地,通过继承 WebChromeClient 并重载它的方法也可以实现不同功能的定制,常用方法及其功能如表 11-5 所示。

表 11-5 WebChromeClient 的常用方法

方 法 名	功 能 说 明
public void onProgressChanged (WebView view, int newProgress)	获得网页的加载进度。参数 newProgress 是当前页面载入进度,取值为 0~100 的整数
public void onReceivedTitle(WebView view, String title)	获取 Web 页中的 title 用来设置自己界面中的 title,当加载出错的时候,该方法获取的标题为“找不到该网页”
public void onReceivedIcon(WebView view, Bitmap icon);	当前页面有个新的图标时,会回调这个方法,获取 Web 页中的 icon
public boolean onCreateWindow (WebView view, boolean isDialog, boolean isUserGesture, Message resultMsg)	请求主机应用创建一个新窗口。如果主机应用选择响应这个请求,则该方法返回 true,并创建一个新的 WebView,将其插入到视图系统中,并将其提供的 resultMsg 作为参数提供给新的 WebView。如果主机应用选择不响应这个请求,则该方法返回 false。默认情况下,该方法不做任何处理并返回 false
public void onCloseWindow(WebView window);	通知主机应用 WebView 已关闭,并在需要的时候从 view 系统中移除它。此时,WebCore 已经停止窗口中的所有加载进度,并在 JavaScript 中移除了所有 cross-scripting 的功能。参数 window 为需要关闭的 WebView

续表

方 法 名	功 能 说 明
public boolean onJsAlert (WebView view, String url, String message, JsResult result);	通知应用程序显示 JavaScript alert 对话框。如果应用程序返回 true,内核认为应用程序处理这个消息;如果返回 false,内核自己处理
public boolean onJsPrompt (WebView view, String url, String message, String defaultValue, JsPromptResult result)	通知应用程序显示一个 prompt 对话框。如果应用程序返回 true,内核认为应用程序处理这个消息;如果返回 false,内核自己处理
public boolean onJsConfirm (WebView view, String url, String message, JsResult result);	通知应用程序显示 JavaScript Confirm 对话框。如果应用程序返回 true 内核认为应用程序处理这个消息;如果返回 false,内核自己处理

11.3.3 WebView 应用实例

【例 11-6】 工程 Demo_11_WebView 演示了在 Activity 中嵌入 WebView 的用法。首先定义布局文件 activity_main.xml,其内容如代码段 11-14 所示。

代码段 11-14 界面布局

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="WebView 示例" />
    <WebView
        android:id="@+id/webview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

定义 MainActivity.java 文件,重写 onCreate()方法,调用 findViewById()方法获得 WebView 的实例对象。然后调用 getSettings()方法取得一个 WebSettings 对象,将 WebView 的 JavaScript 设置成可用。如果加载到 WebView 中的网页使用了 JavaScript,就需要在 WebSettings 中开启对 JavaScript 的支持,因为 WebView 中默认的是 JavaScript 未启用。

最后,调用 `loadUrl(String)` 加载一个网页,如代码段 11-15 所示。

代码段 11-15 加载网页

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.setTitle("WebView 示例");
    mywebview = (WebView) findViewById(R.id.webview);
    //获取 WebView 控件实例
    mywebview.getSettings().setJavaScriptEnabled(true);
    //设置 WebView 属性,使其能够执行 JavaScript 脚本
    mywebview.loadUrl("http://m.baidu.com/");
    //加载需要显示的网页
}
```

在 `MainActivity` 中添加一个继承自 `WebViewClient` 的内部类 `HelloWebViewClient`,如代码段 11-16 所示。其作用是启用 Activity 处理自己的 URL 请求。否则,当点击网页中的一个链接时,默认的 Android 浏览器会处理这个 Intent 来显示一个网页,而不是由 Activity 自己来处理。

代码段 11-16 定义继承自 `WebViewClient` 的内部类

```
private class HelloWebViewClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        view.loadUrl(url);
        return true;
    }
}
```

`WebView` 对象初始化之后,为 `WebViewClient` 设置一个 `HelloWebViewClient` 的实例。

```
mywebview.setWebViewClient(new HelloWebViewClient()); //设置 Web 视图
```

本例中重写了 Activity 类的 `onKeyDown()` 方法。用 `WebView` 显示网页,如果不做任何处理,按设备的“返回”键,整个浏览器会调用 `finish()` 方法结束自身,而不是回退到上一页面。为了让 `WebView` 支持回退功能,需要重写 Activity 类的 `onKeyDown()` 方法,在此方法中处理 Back 事件。如代码段 11-17 所示。

代码段 11-17 重写 `onKeyDown()` 方法

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_BACK) && mywebview.canGoBack()) {
        mywebview.goBack(); //返回 WebView 的上一页面
    }
}
```

```
        return true;
    }
    return false;
}
```

onKeyDown(int, KeyEvent)回调方法将会在 Activity 中按键被按下的时候被调用。当按下的键是 BACK 键并且 WebView 可以回退,即它有历史记录时,就会调用 goBack()方法在 WebView 历史中回退一步。返回 true 表明这个事件已经被处理了。如果条件不满足,这个事件就会被回送给系统。

示例程序的运行结果如图 11-4 所示。



图 11-4 在 Activity 中嵌入 WebView

11.4 本章小结

本章介绍了 Web 应用程序的相关技术和设计方法。利用 URLConnection、HttpURLConnection 或 Socket 可以实现与远程服务器的通信和交互,获取网络中的各种资源;在 Activity 中嵌入 WebView 可以显示从服务器端下载的网页数据。本章学习的重点是网络通信的原理和方法。

习 题

1. 使用 `HttpURLConnection` 从 Internet 上获取一个图片资源,并将其显示在 Activity 中。
2. 设计一个利用 Socket 通信的程序,要求建立连接后,ClientSocket 向 ServerSocket 发送字符串“Hello, This is Socket001.”,服务器端接收到这个字符串后,将其打印到控制台。
3. 设计一个利用 Socket 通信的程序,要求建立连接后,ClientSocket 向 ServerSocket 发送英文字符串,服务器端接收到这个字符串后将其转换为大写字母再传回,客户端接收到返回的字符串后将其显示到 Activity 中。
4. 编写一个可以发送和接收文本内容的简易聊天程序。
5. 在示例工程 Demo_11_WebView 的基础上,增加一个文本框用于输入网址,增加“前进”“后退”“转到”3 个按钮,分别实现网页按照历史记录向前、向后跳转,以及按照文本框中输入的网址直接跳转。

第 12 章

综合应用实例

本章介绍两个综合应用的实例,通过学习这些实例,加深对基本知识的理解,提高 Android 系统各个功能综合应用的能力。

12.1 计算器 APP

【例 12 1】 示例工程 Demo_12_Calculator 实现了一个自定义的计算器程序,实现整数和小数的加减乘除运算。

工程中使用了 UI 界面控件、菜单、对话框、提示信息等,涉及的知识点包括 XML 布局文件的设计、9. patch 格式图片的应用、对按钮点击事件的捕获与响应、基于 SharedPreferences 的数据存取、文本文件的读取、菜单和子菜单的设计和实现、对话框 AlertDialog 的应用、在 AlertDialog 对话框中加载布局、Toast 提示信息的应用等。

12.1.1 功能分析

本例实现一个计算器 APP,实现的计算功能是整数和小数的加减乘除。程序只允许使用界面中提供的按键,包括 0~9 数字键、小数点键、括号键、加减乘除运算符输入键、清零键、删除键以及输出结果的“=”键。这些按键以外的字符全部是非法字符。

按照常规计算器的布局,界面上部是输入和输出区域,下部是功能按钮区域。输入和输出区域不显示光标,没有焦点。文字包括两行,第二行文字较大,实时显示按键生成的计算式。当用户按下“=”键时,显示两行文字,第一行文字较小,显示用户生成的计算式,第二行文字较大,显示运算结果。当输入的算式不合法时,文本框给出错误提示。

按下“清零”键,显示区域显示 0;按下“删除”键,删除最后一次输入的数字或运算符。按下数字键、小数点键、括号键或加减乘除键,则在文本框中实时回显生成的算式。

12.1.2 界面布局设计

1. 准备 9. png 图片文件

为了使程序界面更美观,本例使用 ImageButton 控件实现功能按钮。在设计程序之前需要准备 18 个“.9. png”图片文件,图片中的内容分别是数字和运算符号。图片文件

放置到 drawable 文件夹中。

“.9.png”是 Android 平台应用软件开发使用的一种特殊的图片格式。Android 平台有多种不同的分辨率,很多设备还能自动切换横屏和竖屏,这就导致很多控件的贴图文件会被放大拉伸,或因为长宽的变化而产生拉伸,造成图形的失真变形、边角模糊。使用 “.9.png”技术,可以将图片横向和纵向同时进行拉伸,以实现在多分辨率下仍能保留图像的渐变质感和边角的精细度。

这种技术相当于把一张 png 图片分成了 9 个部分,分别为 4 个角、4 条边以及一个中间区域,4 个角是不做拉伸的,所以拉伸时可以一直保持边角的清晰状态。也可以使用 Android 提供的 draw9patch 工具(路径为 Android/sdk/tools/draw9patch.bat)编辑 “.9.png”图片,其用户界面如图 12-1 所示。也可以用图像处理工具(如 Photoshop)将一个已有的.png 图片编辑成 “.9.png”图片。

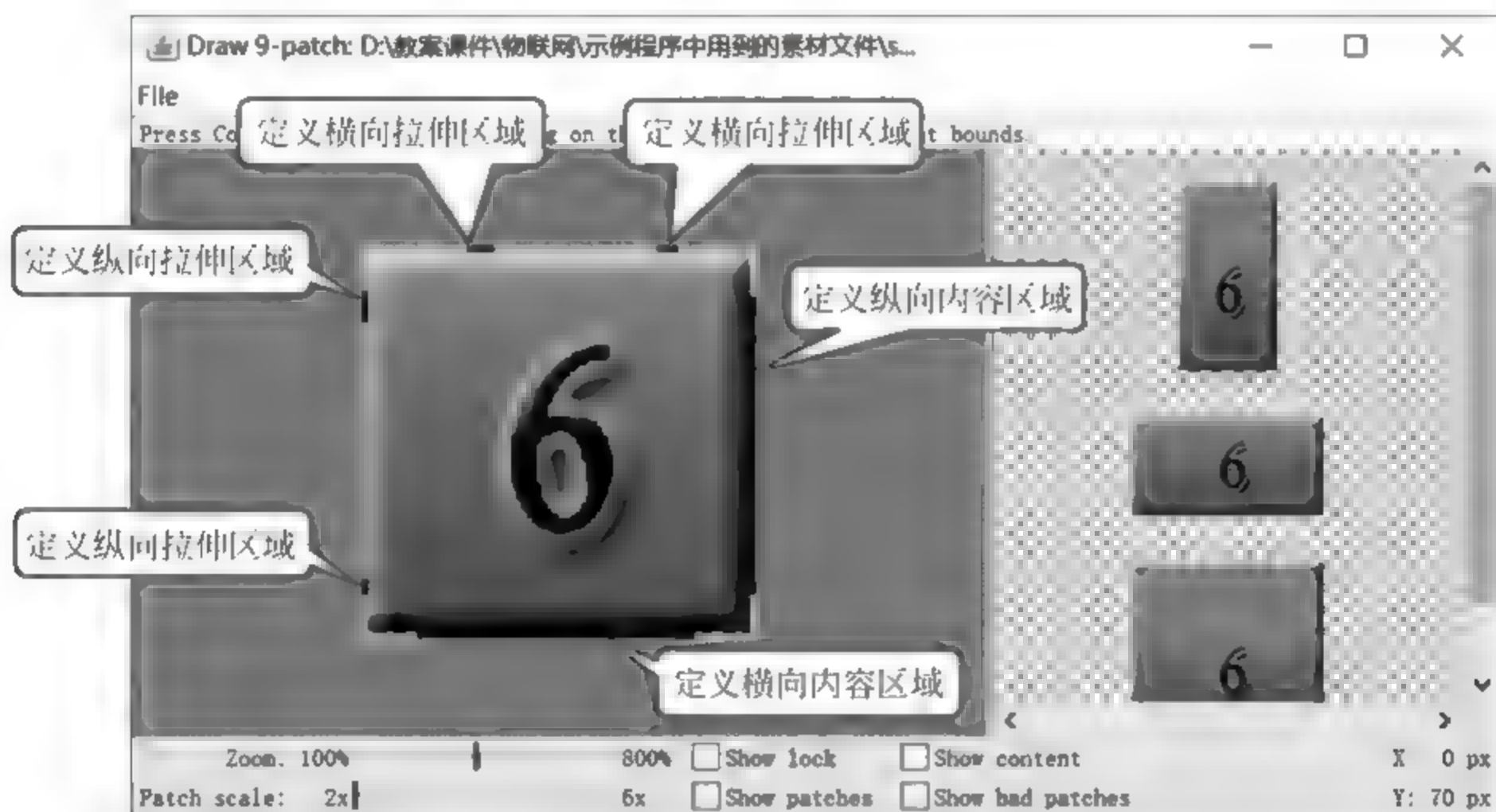


图 12-1 draw9patch 工具的界面

如图 12-1 所示,可以将图片最上侧 1px 边框中的一个或多个点设置为黑色,这些黑色的点定义了图片中可以被横向拉伸的区域。同样也可以将图片最左侧 1px 边框中的一个或多个点设置为黑色,这些黑色的点定义了图片中可以被纵向拉伸的区域。横向拉伸像素点与纵向拉伸像素点相交定义了图片中可拉伸的矩形区域,这样就实现了对图片中一部分区域进行拉伸。

可以选择性地对图片的底边和右边设置黑色线段,用这些黑色线段定义图片的内容区域。当图片作为 UI 控件的背景时,定义其内容区域很重要,控件中的内容(例如文本)都会放到内容区域中。将图片最下侧 1px 边框设置一条黑色线段,该横向线段定义了图片的横向内容区域。将图片最右侧 1px 边框设置一条黑色线段,该纵向线段定义了图片的纵向内容区域。横向线段与纵向线段组成的矩形区域就是内容区域。如果不定义图片的内容区域,那么图片的内容区域就是整个图片区域。

“.9.png”最外侧四边中的像素要么是纯透明、纯白色,要么是纯黑色,不能设置其他颜色和透明度。

draw9patch 工具窗口中,通过鼠标单击可以将最外层中的像素设置为黑色,按住 Shift 键再单击黑色像素可以将黑色像素重置为透明。在左侧窗格的编辑会实时在右侧预览区中显示出拉伸的效果。右侧预览区中有 3 个图片,第一个图片表示的是垂直方向进行拉伸的预览效果图,第二个图片表示的是水平方向进行拉伸的预览效果图,第三个图片表示的是同时在水平和垂直方向上进行拉伸的预览效果图。

2. 设计 Activity 的界面布局

本例 Activity 的界面布局如图 12-2 所示。界面采用嵌套的 LinearLayout 布局,最外层的 LinearLayout 采用垂直布局,包含 6 个子布局。这 6 个子布局也是 LinearLayout 布局,除第一个以外均采用水平布局。第一个子布局包含一个 TextView 和一个 EditText,用于显示按键和计算的结果。其余的 5 个 LinearLayout 控制 18 个按钮的布局。为使软件能适应不同分辨率的移动设备,所有按钮的 layout_width 和 layout_height 属性都设为 match_parent,而控制按钮的大小通过设置 layout_weight 属性值来实现。这样做的好处是控件的大小只和屏幕大小和控件占屏幕的比例有关。



图 12-2 Activity 的界面

图 12-2 对应的布局文件为 res/layout/activity_white.xml 文件,内容如代码段 12-1 所示。

代码段 12-1 界面布局

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1.5">
        <TextView android:layout_width="match_parent"
            android:layout_height="0dp"
            android:id="@+id/editText2"
            style="@style/LittleNumberStyle_Calculator_3"
            android:layout_weight="2"/>
        <EditText android:layout_width="match_parent"
            android:layout_height="0dp"
            android:id="@+id/editText"
            style="@style/NumberStyle_Calculator_3"
            android:layout_weight="5"
            android:text="0"/>
    </LinearLayout>
    <LinearLayout android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1">
        <Button
            android:id="@+id/button_clean"
            android:layout_height="match_parent"
            android:layout_width="match_parent"
            android:layout_weight="1"
            android:text="清零"
            style="@style/BtnStyle_Calculator_3"/>
        <Button
            android:id="@+id/button_left"
            android:layout_height="match_parent"
            android:layout_width="match_parent"
            android:layout_weight="1"
            android:text="("
            style="@style/BtnStyle_Calculator_3"/>
        <Button
            android:id="@+id/button_right"
```

```

        android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:layout_weight="1"
        android:text=")"
        style="@style/BtnStyle_Calculator_3"/>
    <Button
        android:id="@+id/button_delete"
        android:layout_height="match_parent"
        android:layout_width="match_parent"
        android:layout_weight="1"
        android:text="回退"
        style="@style/BtnStyle_Calculator_3"/>
</LinearLayout>
<!-- 其余按钮的布局代码与此类似,略-->
</LinearLayout>

```

12.1.3 实现运算的类

在工程中新建一个类文件 Calculate.java,该类的功能是计算用字符串表示的表达式的值。

本例利用堆栈处理用字符串表示的计算式,其基本过程是:首先创建两个堆栈,一个用来放数字(numStack),另一个用来放运算符(chStack);然后读取运算式,将相应的字符转换为正确的数据格式,压入堆栈。

压栈的过程是从左到右读入算术式,如果读到的是数字,则压入(push)到 numStack 栈中。若读到的是运算符,则先判断 chStack 栈顶元素,若栈顶元素优先级大于读到的运算符,则先将栈顶元素和 numStack 中两个数拿出来计算,再将读到的运算符压入 chStack 中,若读到的运算符优先级大于栈顶元素,则将读到的运算符压入 chStack 中。

如果读到了运算式的最后,则将两个堆栈中的内容全拿出来计算,最后结果放在 numStack 中。加号和减号的优先级较低,乘号和除号的优先级较高。因为用到了堆栈,需要在代码之前使用 import 语句引入 java.util.Stack。Calculate 类的代码如代码段 12-2 所示。

代码段 12-2 Calculate 类的代码

```

//package 和 import 语句略
public class Calculate {
    private Stack<Character> chStack;           //创建一个符号栈
    private Stack<Double> numStack;            //创建一个数字栈
    private StringBuffer expression;
    //功能:初始化表达式
    public Calculate(String expression) {
        this.expression = new StringBuffer(expression);
        //复制 expression 的内容
    }
}

```

```
this.chStack = new Stack<Character> ();
this.numStack = new Stack<Double> ();
}
//功能:计算表达式的值
public double result() throws Exception {
    //若表达式还没有解析完
    while (this.expression.length() > 0) {
        //获取当前表达式头部的第一个字符
        char ch = this.expression.charAt(0);
        this.expression.deleteCharAt(0);           //删除第一个字符(取一个,删除一个)
        double num = 0;
        boolean existNum = false;
        //若当前读取到的是数字
        while (ch >= '0' && ch <= '9') {
            num = num * 10 + ch - '0';
            //减零是为了使 ch 表示实际的数值,而不是 ASCII 码值
        }
        existNum = true;
        //继续取数
        if (this.expression.length() > 0) {
            ch = this.expression.charAt(0);
            this.expression.deleteCharAt(0);
        } else {
            break;
        }
    }
    if (ch == '.') {
        ch = this.expression.charAt(0);
        this.expression.deleteCharAt(0);
        int i = 1;
        while (ch >= '0' && ch <= '9') {
            double m1 = Math.pow(0.1, i);
            i++;
            num = num + (ch - '0') * m1;
            existNum = true;
            //继续取数
            if (this.expression.length() > 0) {
                ch = this.expression.charAt(0);
                this.expression.deleteCharAt(0);
            } else {
                break;
            }
        }
    }
}
```

```

//若刚刚解析完一个数字,则将数字压栈
if (existNum) {
    this.numStack.push(num);
    //若整个表达式的解析已经结束了,这种情况为以数字结束
    if (this.expression.length() == 0 && ch > '0' && ch <= '9') {
        break; //结束 while 循环
    }
}
//若符号栈为空,或栈顶为左括号,或 ch 本身就是左括号,则直接将符号压入栈
if (this.chStack.isEmpty() || this.chStack.peek() == '(' || ch == '(') {
    this.chStack.push(ch);
    continue;
}
switch (ch) {
    case ')': {
        //若当前符号是右括号,则不断弹出一个运算符和两个操作数,直到遇到
        //左括号为止
        while (this.numStack.size() >= 2 && !this.chStack.isEmpty()
            && this.chStack.peek() != '(') {
            this.calc();
        }
        if (!this.chStack.isEmpty() && this.chStack.peek() == '(') {
            this.chStack.pop(); //弹出这个左括号
            continue;
        } else {
            throw new IllegalArgumentException("括号的数量不匹配!");
        }
    }
    case '*':
    case '/': {
        //若符号栈栈顶元素为+、-、( 或者符号栈为空,则意味着符号栈栈顶符
        //号比 ch 优先级低,所以,将 ch 压栈。否则,将符号栈栈顶元素弹出
        //来,然后开始计算
        while (this.numStack.size() >= 2 && !(this.chStack.isEmpty()
            || this.chStack.peek() == '(' || this.chStack.peek() == '+'
            || this.chStack.peek() == '-')) {
            this.calc();
        }
        //若符号栈栈顶元素优先级比 ch 的低
        if (this.chStack.isEmpty() || this.chStack.peek() == '('
            || this.chStack.peek() == '+'
            || this.chStack.peek() == '-') {

```

```

        this.chStack.push(ch);
        continue;
    }
}
case '+':
case '=': {
    //若当前符号栈栈顶元素不是'(',符号栈也不为空,则将符号栈栈顶元
    //素弹出来,然后开始计算。因为+、号的优先级最低
    while (this.numStack.size() >= 2 && (this.chStack.peek() == '*'
        || this.chStack.peek() == '/'
        || this.chStack.peek() != '(')) {
        this.calc();
    }
    if (this.chStack.isEmpty()
        || this.chStack.peek() == '('
        || this.chStack.peek() == '+'
        || this.chStack.peek() == '-') {
        //若符号栈栈顶元素为'(',或符号栈为空,则将 ch 压栈
        this.chStack.push(ch);
        continue;
    } else {
        throw new IllegalArgumentException("表达式格式不合法!");
    }
}
default : throw new IllegalArgumentException("运算符非法!");
} //switch 结束
} //while 结束
//若符号栈不为空,则不断地从符号栈和数字栈中弹出元素,进行计算
while(!this.chStack.isEmpty()) {
    this.calc();
}
//若最终数字栈中仅存一个元素,则证明表达式正确,栈顶元素就是表达式的值
return this.numStack.size() == 1 ? this.numStack.pop() : null;
}
//功能:依据指定的操作数、运算符进行运算
private void calc() throws Exception {
    double b = this.numStack.pop(); //取出第一个数
    double a = this.numStack.pop(); //取数第二个数
    char op = this.chStack.pop();
    double result = 0;
    switch (op) {
        case '+':
            result = a + b; break;

```

```

        case ' ':
            result = a/b; break;
        case '*':
            result = a * b; break;
        case '/':
            if (b == 0) {
                throw new ArithmeticException("除数不能为 0!");
            }
            result = a/b;
            break;
    }
    //将运算的结果压栈
    this.numStack.push(result);
}
}

```

12.1.4 界面功能的实现

MainActivity 实现计算器程序的主界面,该类继承自 Activity 类,同时实现了 OnClickListener 接口。类中设置了一个字符串变量 tem,用于暂存输入的计算式。当用户按“=”键时,将依据这个字符串的内容进行计算。同时它也是计算器的输入输出区域中显示出来的计算式。

首先重写 onCreate() 方法,实例化布局中的各控件。接下来对各个键绑定监听器,实现算术式的输入功能和计算输出算术式值的功能。“清零”键、“回退”键、等号键的功能较特殊,需要单独分别处理。其他的键作为基本算式的输入键,可看作一类,处理方式类似。

1. “清零”键

“清零”键的功能是清空输入和输出区域中的内容,其点击事件的主要处理如代码段 12-3 所示。

代码段 12-3 处理“清零”键的点击事件

```

public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button_clean:
            editText.setText("0");
            tvEquation.setText("");
            tem = "";
            ifEqu = false;
            break;
    }
}
}

```

2. “回退”键

“回退”键的功能是删除最后一次输入的数字,即当前表达式的最后一个字符,其点击事件的主要处理如代码段 12-4 所示。

代码段 12-4 处理“删除”按钮的点击事件

```
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button_delete:
            if (tem.length() == 0 || tem.length() == 1) {
                //edittext1 中没有任何数据,或只有一个数或运算符
                edittext.setText("0");
            }
            else {
                tem = tem.substring(0, tem.length() - 1); //删除最后一个字符
                edittext.setText(tem);
            }
            ifEqu = false;
            break;
    }
}
```

3. “=”键

“=”键的功能是计算输入算式的值,并将结果显示在文本框中,同时将算术式显示在文本框上方的 TextView 控件中。其点击事件的主要处理如代码段 12-5 所示。

代码段 12-5 处理“=”键的点击事件

```
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button_equ:
            str_calculate = edittext.getText().toString(); //获得输入的算式
            Calculate ep = new Calculate(str_calculate); //计算表达式的值
            try {
                double result = ep.result();
                String result_str = String.valueOf(result);
                tvEquation.setText(str_calculate + "=");
                edittext.setText(result_str); //显示结果
                tem = result_str;
            } catch (Exception e) {
                //TODO Auto-generated catch block
                e.printStackTrace();
                tvEquation.setText(str_calculate + "");
            }
    }
}
```

```

        editText.setText("非法的输入算式!");
    }
    ifEqu=true;
    break;
}
}

```

4. 其他键

如果按数字或运算符键,则根据按键的内容在字符串 tem 的末尾增加相应的字符,同时将字符串显示在输出区域。例如,当按“0”键时的处理如代码段 12-6 所示。

代码段 12-6 处理“0”键的点击事件

```

public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button0:
            handleInputNumber(0);
            break;
    }
}

private void handleInputNumber(int n) { //处理点击数字键
    if (ifEqu==true) { //上一个点击的键是等号键,下一个数字重新开始
        tem="";
    }
    this.firstzero();
    tem = tem + n;
    editText.setText(tem);
    ifEqu= false;
}

```

其中,firstzero()方法用于处理当数字的第一个字符为0的情况,如果出现这种情况,并且0后面不是小数点,这个输入的0就不会计入算术式中。该方法的定义如代码段 12-7 所示。

代码段 12-7 firstzero()方法的定义

```

public void firstzero() {
    if(tem.length()>1) {
        int sum1=tem.length()-1;
        //在加、减、乘、除之后,若输入的是“0”,而且再输入的是数字,则 tem 不会增加字符
        if ((tem.charAt(sum1-1)=='+'||tem.charAt(sum1-1)=='-'||
            tem.charAt(sum1-1)=='*'||tem.charAt(sum1-1)=='/'||)
            &&tem.charAt(sum1)=='0'){
            tem=tem.substring(0,sum1);
        }
    }
}

```

5. 设计界面的容错功能

为了增强应用程序的可用性,对于算术式输入键要设置一定的容错功能,以避免生成非法的算术式。本例中设置的容错控制包括:不能连续输入两个小数点,不能连续输入两个运算符,第一个输入的不能是+、-、*、÷、小数点等。当出现上述情况时,输入的内容不会被添加到算术式中,并会弹出一个 Toast 提示信息来提醒用户。具体代码不再赘述,详见随书源程序。

12.1.5 实现基于 SharedPreferences 的数据存取

在工程中新建一个类文件 PreferencesService.java,该类的功能是实现配置参数的存取,参数采用 SharedPreferences 方式存储,文件名为 skin_file.xml。

PreferencesService 类的代码如代码段 12-8 所示。

代码段 12-8 Calculate 类的代码

//package 和 import 语句略

```
public class PreferencesService {  
    private Context context;  
    public PreferencesService(Context context) {  
        super();  
        this.context = context;  
    }  
    public void save(int skin) { //存储 APP 皮肤参数  
        //首先取得 SharedPreferences 类型的对象  
        SharedPreferences preferences=context.getSharedPreferences("skin_  
            file", Context.MODE_PRIVATE);  
        //参数 1:指定 XML 文件的名称,参数 2:文件的操作模式,不允许其他应用访问此文件  
        Editor editor=preferences.edit();  
        editor.putInt("skin", skin);  
        editor.commit(); //将数据提交到 XML 文件中  
    }  
    public Map<String,String>getPreferences() { //获取配置参数  
        Map<String,String>params=new HashMap<String,String>();  
        SharedPreferences preferences=context.getSharedPreferences("skin_  
            file", Context.MODE_PRIVATE);  
        params.put("skin", String.valueOf(preferences.getInt("skin", 0)));  
        return params;  
    }  
}
```

12.1.6 菜单设计

本例使用 Menu 菜单实现更换皮肤、查看帮助信息、查看版权信息以及退出的功能,如图 12-3 所示。



图 12-3 计算器 APP 的菜单和子菜单

菜单采用 XML 方式实现。先在 res/menu 文件夹中新建 menu.xml 文件,在其中添加菜单项,相关代码如代码段 12-9 所示。

代码段 12-9 菜单资源文件

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/skin"
        android:title="更换皮肤">
        <menu>
            <item
                android:id="@+id/skin_black"
                android:title="诱惑黑"/>
            <item
                android:id="@+id/skin_purple"
                android:title="浪漫紫"/>
            <item
                android:id="@+id/skin_white"
                android:title="简约白"/>
        </menu>
    </item>
    <item android:id="@+id/help_dialog"
        android:title="帮助"/>
    <item android:id="@+id/about"
        android:title="关于"/>
    <item android:id="@+id/exit"
        android:title="退出"/>
</menu>
```

重写 MainActivity 中的 onCreateOptionsMenu() 方法, 在界面中添加菜单。本例中调用 inflate() 方法生成菜单, 该方法使用一个指定的 XML 资源填充菜单, 这里指定的是前一步骤创建的 menu.xml 文件。如果出现错误, 该方法会抛出 InflateException 异常信息。相关代码如代码段 12-10 所示。

代码段 12-10 添加菜单

```
public boolean onCreateOptionsMenu(Menu menu) {  
    MenuInflater inflater = getMenuInflater();           //获得 menu 容器  
    inflater.inflate(R.menu.menu, menu);                //用 menu.xml 填充 menu 容器  
    return super.onCreateOptionsMenu(menu);  
}
```

选择“更换皮肤”菜单项下的子菜单, 则加载相应的布局文件, 实现界面风格的切换。选择“帮助”菜单项, 则创建并显示帮助对话框, 如图 12-4 所示。帮助信息存储在文本文件中。为了提高程序的可维护性, 程序读出帮助文件的内容并将其显示在对话框中。

选择“退出”菜单项弹出确认退出对话框, 如图 12-5 所示。选择“关于”菜单项显示计算器 APP 版权信息对话框。

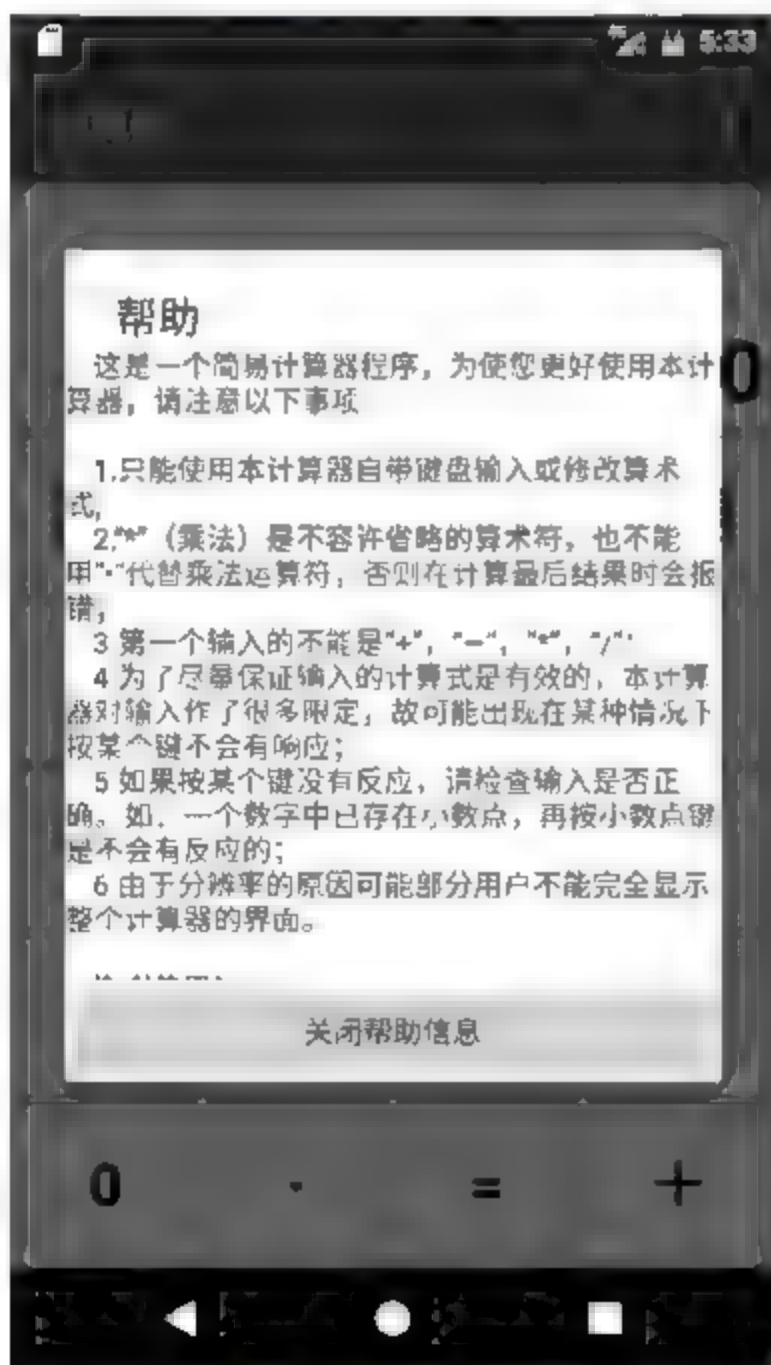


图 12-4 帮助对话框

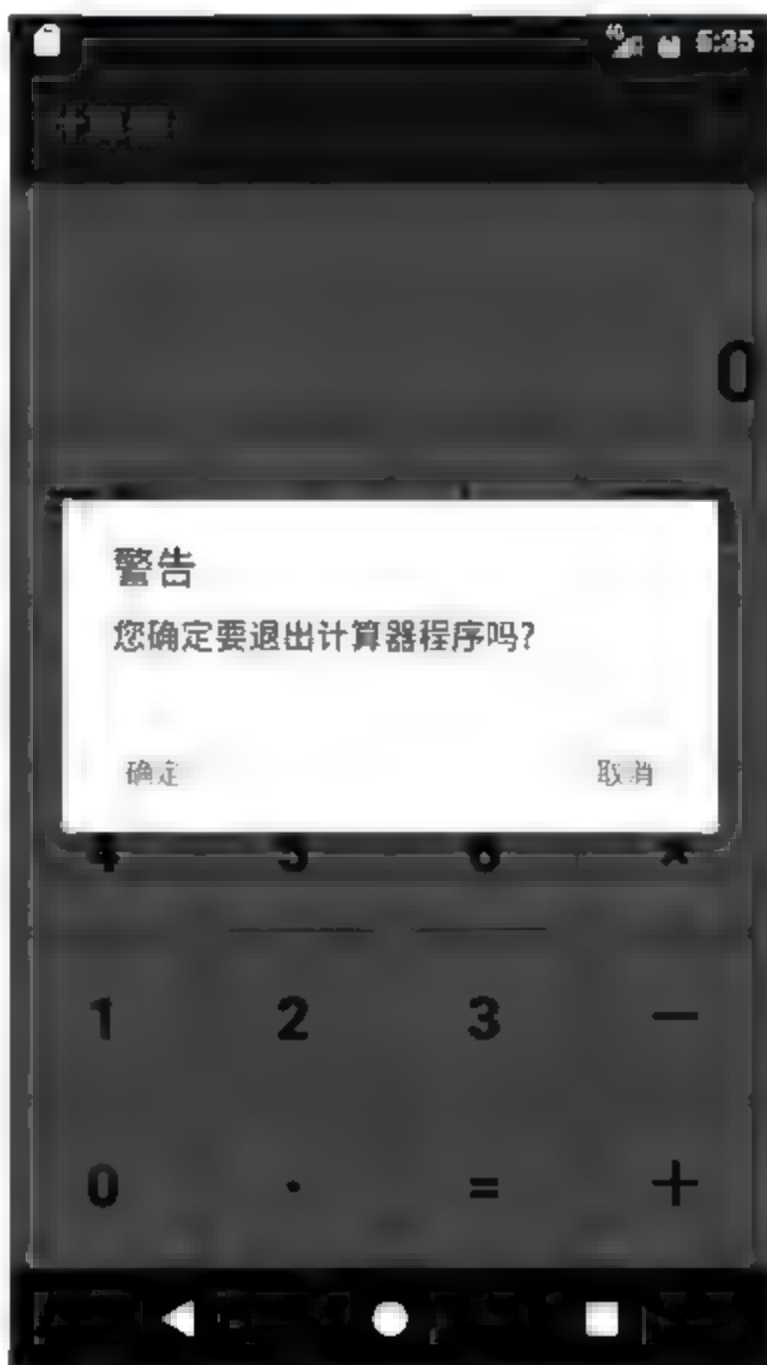


图 12-5 确认退出对话框

重写 onOptionsItemSelected(MenuItem item) 方法, 实现各个菜单项的功能, 如代码段 12-11 所示。

代码段 12-11 实现各个菜单项的功能

```

service = new PreferencesService(MainActivity.this);
//先读取以前保存的参数值
Map<String,String> params = service.getPreferences();
skin = Integer.parseInt(params.get("skin"));
public boolean onOptionsItemSelected(MenuItem item) {
    if (item.getItemId() == R.id.skin_black) {
        //更换计算器的外观皮肤,黑色外观
        setContentView(R.layout.activity_black);
        getView();
        editText.setText(tem); //更换皮肤时,保证文本框中的文字不变化
        tvEquation.setText(str_calculate + "=");
        //更换皮肤时,保证文本框中的文字不变化
        service.save(0); //使用 SharedPreferences 保存用户的配置参数
    }
    if (item.getItemId() == R.id.skin_purple) {
        //更换计算器的外观皮肤,紫色外观
        setContentView(R.layout.activity_purple);
        getView();
        editText.setText(tem);
        tvEquation.setText(str_calculate + "=");
        service.save(1); //使用 SharedPreferences 保存用户的配置参数
    }
    if (item.getItemId() == R.id.skin_white) {
        //更换计算器的外观皮肤,白色外观
        setContentView(R.layout.activity_white);
        getView();
        editText.setText(tem);
        tvEquation.setText(str_calculate + "=");
        service.save(2); //使用 SharedPreferences 保存用户的配置参数
    }
    if (item.getItemId() == R.id.help_dialog) {
        //显示帮助对话框
        AlertDialog.Builder helpAlertBuilder = new Builder(MainActivity.this);
        helpAlertBuilder.setTitle("帮助");
        LayoutInflater inflater = LayoutInflater.from(MainActivity.this);
        View helpAllView = inflater.inflate(R.layout.help_dialog, null);
        Button bt_hlp = (Button) helpAllView.findViewById(R.id.helpButton);
        TextView helpTextView = (TextView) helpAllView.findViewById(R.id.helpDocView);
    }
}

```

```
String helpText = "";
try {
    InputStream in = getResources().openRawResource(R.raw.help);
    InputStreamReader inputStreamReader = new InputStreamReader
        (in, "UTF 8");
    char myContent[] = new char[in.available()];
    inputStreamReader.read(myContent);
    helpText = new String(myContent);
    inputStreamReader.close();
    in.close();
} catch (Exception e) {
    e.printStackTrace();
}
helpTextView.setText(helpText);
helpAlertBuilder.setView(helpAllView);
//在 AlertDialog 对话框中加载布局
final AlertDialog helpDialog = helpAlertBuilder.create();
bt_hlp.setOnClickListener(new android.view.View.OnClickListener() {
    @Override
    public void onClick(View v) {
        helpDialog.dismiss();
    }
});
helpDialog.show();
}
if (item.getItemId() == R.id.exit) { //exit 功能
    Builder exitAlert = new Builder(MainActivity.this);
    //exitAlert.setIcon(R.drawable.warning);
    exitAlert.setTitle("警告");
    exitAlert.setMessage("您确定要退出计算器程序吗?");
    exitAlert.setNeutralButton("确定", new DialogInterface.
        OnClickListener() {
            public void onClick(DialogInterface arg0, int arg1) {
                MainActivity.this.finish();
            }
        });
    exitAlert.setNegativeButton("取消", new DialogInterface.
        OnClickListener() {
            public void onClick(DialogInterface arg0, int arg1) {
            }
        });
});
```

```
        exitAlert.create();
        exitAlert.show();
    }
    if(item.getItemId()==R.id.about) {           //如果点击的是 about,则弹出对话框
        Builder exitAlert=new Builder(MainActivity.this);
        exitAlert.setTitle("版权声明:");
        exitAlert.setMessage("这是教材的示例程序!\n版本号:2.0");
        exitAlert.setNegativeButton("确定", new DialogInterface.
            OnClickListener() {
                public void onClick(DialogInterface arg0, int arg1) {
                }
            });
        exitAlert.create();
        exitAlert.show();
    }
    return super.onOptionsItemSelected(item);
}
```

12.2 待办事项提醒小助手

【例 12 2】 示例工程 Demo_12_ToDoReminder 实现了一个用于待办事项提醒的 APP 程序。

工程中使用了 UI 界面控件、Fragment、菜单、对话框等。涉及的知识点包括 XML 布局文件的设计、基于 Fragment 的界面切换和参数传递、自定义 ListView 列表项的布局以及利用 SimpleAdapter 实现 ListView 的多列显示、对 ListView 列表项点击和长按事件的捕获和响应、对按钮点击事件的捕获与响应、菜单、子菜单、ActionBar 上的菜单按钮(溢出菜单)、在 AlertDialog 对话框中加载布局、日期和时间选择对话框的使用、基于 SQLite 数据库的数据存取、文本文件的读取、Notification 消息的定时推送等。

12.2.1 功能分析

本例实现一个用于待办事项提醒的 APP 程序。该程序的主界面按时间顺序列出今天、明天、后天以及之后的待办事项。在程序中可以添加、修改和删除待办事项,可以设置每个待办事项的提醒时间。当预设的待办事项提醒时间到时,利用 Notification 在状态栏弹出提醒消息。

程序中使用 SQLite 数据库存储待办事项的日期和内容。每一项待办事项有一个唯一的 ID 号标识。打开应用程序,主页面按照今天、明天、后天的顺序列出全部提醒。当设定的时间到时,会弹出 Notification 提醒消息。

12.2.2 创建数据库

新建一个类 `MyDBOpenHelper`, 继承自 `SQLiteOpenHelper`。重写其构造方法和 `onCreate()` 方法。数据库文件存储在 `/data/data/edu.hebust.xxy.demo_12_todoreminder/databases` 目录中, 数据库名称为 `todoDatabase.db`, 如图 12-6 所示。

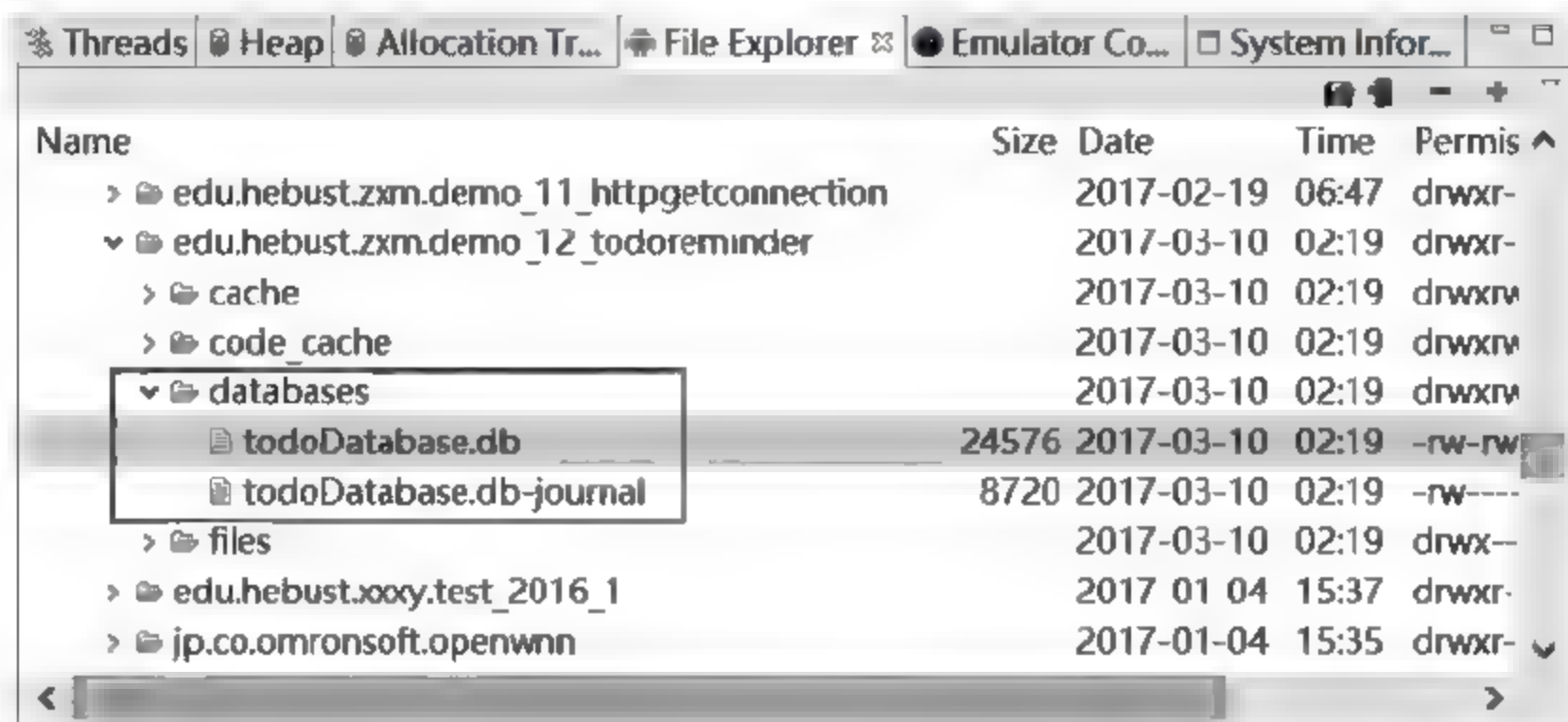


图 12-6 数据库文件

数据表 `tb_ToDoItem` 用于存储待办事项信息, 其结构如表 12-1 所示。本例使用 `SimpleAdapter` 适配器将数据表中的数据绑定到 `ListView` 控件中。

表 12-1 数据表 `tb_ToDoItem` 的结构

列 名	数 据 类 型	说 明
<code>_id</code>	<code>integer</code>	每个待办事项的 ID, 主键, 自动增加
<code>remindTitle</code>	<code>text</code>	待办事项的标题, 不能为 <code>null</code>
<code>createDate</code>	<code>text</code>	待办事项的创建日期和时间
<code>modified</code>	<code>boolean</code>	是否曾经修改, 默认值为 <code>false</code>
<code>modifyDate</code>	<code>text</code>	最后修改的日期和时间
<code>remindText</code>	<code>text</code>	待办事项的注释说明
<code>remindDate</code>	<code>text</code>	待办事项的提醒日期和时间
<code>haveDo</code>	<code>boolean</code>	待办事项的处理状态, 默认值为 <code>false</code>

`MyDBOpenHelper` 类的主要代码如代码段 12-12 所示。实例化这个类, 就可以创建相应的数据库和数据表。

代码段 12-12 定义 `SQLiteOpenHelper`

```
//package 和 import 语句略
public class MyDBOpenHelper extends SQLiteOpenHelper {
    public MyDBOpenHelper(Context context) {
```

```
//重写构造方法,创建一个名为 DB ToDoList 的数据库
super(context, "DB ToDoList", null, 1);
}
@Override
public void onCreate(SQLiteDatabase db) {
    //重写 onCreate() 方法,创建数据表,其中 ID 字段作为主键,自动增加
    String sql = "create table tb_ToDoItem(
        _id integer primary key autoincrement, " + //每个待办事项的 ID
        "remindTitle text not null, " + //待办事项的标题文本
        "createDate text, " + //待办事项的创建日期和时间
        "modified boolean DEFAULT(0), " + //是否已修改,默认值为 false
        "modifyDate text, " + //最后修改日期和时间
        "remindText text, " + //待办事项的注释说明
        "remindDate text, " + //待办事项的提醒日期和时间
        "haveDo boolean DEFAULT(0));"; //是否已处理,默认值为 false
    db.execSQL(sql); //执行 SQL 语句
}
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    //重写其 onUpgrade 方法
    _db.execSQL("DROP TABLE IF EXISTS tb_ToDoItem");
    onCreate(_db)
}
}
```

12.2.3 界面设计和功能实现

为了实现程序的功能,本例定义了启动界面 MainActivity 类和 7 个 Fragment 类,其类名和相应的功能如表 12-2 所示。MainActivity 通过加载这些 Fragment 实现相应的用户界面及其功能。

表 12-2 Fragment 类及其功能

类 名	功能及其说明
RemindListFragment	主页面,按照待办时间顺序分别列出今天、明天、后天以及之后的待办事项
TodayListFragment	仅显示今日提醒事项
UndoListFragment	仅显示未处理事项
AllListByCreateTimeFragment	按创建时间列出全部提醒事项
AllListByToDoTimeFragment	按待办时间列出全部提醒事项
AddNewFragment	添加新提醒事项
UpdateFragment	修改提醒事项

1. 主页面

RemindListFragment 类用于实现主页面,按照待办时间顺序分别列出今天、明天、后天以及之后的待办事项,包括提醒时间、标题、备注和处理状态,如图 12-7 所示。



图 12-7 主页面的显示效果

页面对应的布局文件如代码段 12-13 所示。

代码段 12-13 主页面布局文件

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/scrollView"
        android:fadingEdge="none"
        android:scrollbars="vertical">
        <LinearLayout
            android:id="@+id/remindLayout"
            android:layout_width="match_parent"
```

```
        android:layout_height="match_parent"
        android:orientation="vertical" >
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal" >
            <TextView
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:textSize="25sp"
                android:padding="10dp"
                android:textColor="#009900"
                android:layout_weight="1"
                android:text="今天:" />
            <TextView
                android:id="@+id/tvToday"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:textSize="15sp"
                android:padding="10dp"
                android:textColor="#ff6600a66"
                android:layout_gravity="right|bottom"
                android:layout_weight="1" />
        </LinearLayout>
        <View
            style="@style/divider_horizontal" />
        <ListView
            android:id="@+id/listToDoToday"
            android:layout_width="match_parent"
            android:layout_height="match_parent"/>
        <View
            style="@style/divider_horizontal" />
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal" >
            <TextView
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:textSize="25sp"
                android:padding="10dp"
                android:textColor="#009900"
                android:layout_weight="1"
```

```
        android:text="明天:"/>
    <TextView
        android:id="@+id/tvTomorrow"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textSize="15sp"
        android:padding="10dp"
        android:textColor="#ff660a66"
        android:layout_gravity="right|bottom"
        android:layout_weight="1" />
</LinearLayout>
<View
    style="@style/divider_horizontal" />
<ListView
    android:id="@+id/listToDoTomorrow"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
<View
    style="@style/divider_horizontal" />
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >
    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textSize="25sp"
        android:padding="10dp"
        android:textColor="#009900"
        android:layout_weight="1"
        android:text="后天:" />
    <TextView
        android:id="@+id/tvAfterTomorrow"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textSize="15sp"
        android:padding="10dp"
        android:textColor="#ff660a66"
        android:layout_gravity="right|bottom"
        android:layout_weight="1" />
</LinearLayout>
<View
    style="@style/divider_horizontal" />
```

```
<ListView
    android:id="@+id/listToDoAfterTomorrow"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
<View
    style="@style/divider_horizontal" />
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >
    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textSize="25sp"
        android:padding="10dp"
        android:textColor="#009900"
        android:layout_weight="1"
        android:text="之后:" />
    <TextView
        android:id="@+id/tvAfterAll"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textSize="15sp"
        android:padding="10dp"
        android:textColor="#ff660a66"
        android:layout_gravity="right|bottom"
        android:layout_weight="1" />
</LinearLayout>
<View
    style="@style/divider_horizontal" />
<ListView
    android:id="@+id/listToDoAfterAll"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
</LinearLayout>
</ScrollView>
</LinearLayout>
```

布局中使用了4个ListView控件,分别用于显示今天、明天、后天以及之后的待办事项列表。为了让这4个ListView同时使用一个滚动条,需要重新设置ListView的高度,这可以通过调用自定义方法 setListViewHeight()实现,该方法的定义如代码段 12-14 所示。

代码段 12-14 重新设置 ListView 高度

```
public static void setListViewHeight (ListView listview) {  
    int totalHeight = 0;  
    ListAdapter adapter=listview.getAdapter();  
    if (null != adapter) {  
        for (int i = 0; i < adapter.getCount(); i++) {  
            View listItem = adapter.getView(i, null, listview);  
            if (null != listItem) {  
                listItem.measure(0, 0);  
                //注意 listview 子项必须为 LinearLayout 才能调用该方法  
                totalHeight += listItem.getMeasuredHeight();  
            }  
        }  
        ViewGroup.LayoutParams params = listview.getLayoutParams();  
        params.height = totalHeight + (listview.getDividerHeight() *  
            (listview.getCount() - 1));  
        listview.setLayoutParams (params);  
    }  
}
```

2. 选择列表项的处理

选择某个列表项,则弹出对话框,显示该提醒项的详细信息,如图 12 8 所示。



图 12 8 选择列表项显示该项的详细信息

对话框设置了3个按钮,分别用于修改提醒事项内容、将提醒事项设置为已处理状态、关闭对话框窗口。选择某个列表项的处理如代码段 12-15 所示。

代码段 12-15 设置选择列表项的响应

```

todoList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                                                    //选择列表项

    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int position,
        long l) {
        HashMap<String, String> temp = (HashMap<String, String>) listViewAdapter.
            getItem(position);
        final String taskID = temp.get("_id");           //获取选择的提醒项 ID
        Cursor result = dbRead.query("tb_ToDoItem", null, "_id=?", new String[]
            {taskID}, null, null, null, null);
        result.moveToFirst();
        HashMap<String, String> itemFindByID = new HashMap<String, String> ();
        itemFindByID.put("id", "ID:" + String.valueOf(result.getInt(0)) + "\n");
        itemFindByID.put("remindTitle", "标题:" + result.getString(1) + "\n");
        itemFindByID.put("createDate", "创建时间:" + result.getString(2) + "\n");
        itemFindByID.put("modified", result.getInt(3) == 0 ? "未修改\n" : "已修改\n");
        itemFindByID.put("modifyDate", "最后修改:" + result.getString(4) + "\n");
        itemFindByID.put("remindText", "备注:" + result.getString(5) + "\n");
        itemFindByID.put("remindDate", "提醒时间:" + result.getString(6) + "\n");
        itemFindByID.put("haveDo", result.getInt(7) == 0 ? "该事项未处理" : "该事
            项已经处理");
        new AlertDialog.Builder(getActivity())
            .setTitle("详细信息")
            .setMessage(itemFindByID.get("id") + itemFindByID.get
                ("remindTitle")
                + itemFindByID.get("createDate") + itemFindByID.get
                ("modified")
                + itemFindByID.get("modifyDate") + itemFindByID.get
                ("remindText")
                + itemFindByID.get("remindDate") + itemFindByID.get
                ("haveDo"))
            .setNegativeButton("设为已处理", new DialogInterface.
                OnClickListener() {
                    public void onClick(DialogInterface arg0, int arg1) {
                        SQLiteDatabase dbWriter = dbOpenHelper.
                            getWritableDatabase();

```

```

        ContentValues cv = new ContentValues();
        cv.put("haveDo", 1);
        dbWriter.update("tb TbDoItem", cv, " id=?", new String[]
            {taskID});
        dbWriter.close();
        getFragmentManager().beginTransaction()
            .replace(R.id.fragment_container, new
                RemindListFragment())
            .commit();
    }
})
.setNeutralButton("修改该项内容", new DialogInterface.
    OnClickListener() {
    public void onClick(DialogInterface arg0, int arg1) {
        SQLiteDatabase dbWriter = dbOpenHelper.
            getWritableDatabase();
        final Bundle bundle = new Bundle();
        bundle.putString("taskID", taskID);
        UpdateFragment updateFragment = new UpdateFragment();
        updateFragment.setArguments(bundle);
        getFragmentManager().beginTransaction()
            .replace(R.id.fragment_container, updateFragment)
            .addToBackStack(null) //为了支持回退键
            .commit();
    }
})
.setPositiveButton("关闭窗口", null)
.create()
.show();
}
});

```

3. 修改提醒项

修改列表中的提醒项,通过 UpdateFragment 类实现,界面如图 12-9 所示。

首先获取用户选择的列表项对应的记录 ID,然后到数据库查询这条记录,逐项显示到界面中的 EditText 控件中,用户修改其中的内容后点击“确定修改”按钮,则将修改的数据提交到数据库。UpdateFragment 类的实现如代码段 12-16 所示。



图 12-9 修改列表项的详细信息

代码段 12-16 UpdateFragment 类的主要代码

//package 和 import 语句略

```
public class UpdateFragment extends Fragment {
    private SQLiteDatabase dbRead;
    private MyDBOpenHelper dbOpenHelper;
    private Button btnUpdate, btnCancel;
    private EditText taskEdit, dateEdit, timeEdit, remarkEdit;
    private TextView taskID;
    private Date remindDate=new Date(System.currentTimeMillis());
    private Calendar newRemindDate=Calendar.getInstance();
    //新版本推荐使用 Calendar,不用 Date
    //初始化必须有,否则产生空指针错误

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_update, container,
            false);
        btnUpdate = (Button) rootView.findViewById(R.id.btnUpdate);
        btnCancel = (Button) rootView.findViewById(R.id.btnUpdateCancel);
        taskID = (TextView) rootView.findViewById(R.id.tvTaskID);
        taskEdit = (EditText) rootView.findViewById(R.id.etUpdateTask);
        dateEdit = (EditText) rootView.findViewById(R.id.etUpdateDate);
    }
}
```

```
timeEdit = (EditText) rootView.findViewById(R.id.etUpdateTime);
remarkEdit = (EditText) rootView.findViewById(R.id.etUpdateRemark);
dbOpenHelper = new MyDBOpenHelper(getActivity().getApplicationContext());
final String updateID = getArguments().getString("taskID");
taskID.setText("ID: " + updateID);
dbRead = dbOpenHelper.getReadableDatabase();
Cursor result = dbRead.query("tb ToDoItem", null, "id=?", new String[]
    {updateID}, null, null, null, null);
result.moveToFirst();
taskEdit.setText(result.getString(1));
final SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy年
    MM月 dd日");
final SimpleDateFormat timeFormatter = new SimpleDateFormat("HH:mm:ss");
final SimpleDateFormat longDateFormatter = new SimpleDateFormat
    ("yyyy-MM-dd HH:mm:ss");
try {
    remindDate = longDateFormatter.parse(result.getString(6));
} catch (ParseException e) {
    e.printStackTrace();
}
dateEdit.setText(dateFormatter.format(remindDate));
timeEdit.setText(timeFormatter.format(remindDate));
remarkEdit.setText(result.getString(5));
newRemindDate.setTime(remindDate);
dateEdit.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        new DatePickerDialog(getActivity(), new DatePickerDialog.
            OnDateSetListener() {
                @Override
                public void onDateSet(DatePicker datePicker, int year, int
                    month, int day) {
                    newRemindDate.set(year, month, day);
                    dateEdit.setText(dateFormatter.format(new Date
                        (newRemindDate.getTimeInMillis())));
                }
            }, newRemindDate.get(Calendar.YEAR), newRemindDate.get
                (Calendar.MONTH), newRemindDate.get(Calendar.DAY_OF_MONTH))
            .show();
    }
});
timeEdit.setOnClickListener(new View.OnClickListener() {
    @Override
```

```

        public void onClick(View view) {
            new TimePickerDialog(getActivity(), new TimePickerDialog.
                OnTimeSetListener() {
                    @Override
                    public void onTimeSet(TimePicker timePicker, int hourOfDay,
                        int minute) {
                        newRemindDate.set(newRemindDate.get(Calendar.YEAR),
                            newRemindDate.get(Calendar.MONTH),
                            newRemindDate.get(Calendar.DAY_OF_MONTH),
                                hourOfDay, minute);
                        timeEdit.setText(timeFormatter.format(
                            new Date(newRemindDate.getTimeInMillis())));
                    }
                }, newRemindDate.get(Calendar.HOUR_OF_DAY),
                    newRemindDate.get(Calendar.MINUTE), true)
                .show();
        }
    });
    btnCancel.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            getFragmentManager().popBackStack(); //回退到上一个界面
        }
    });
    btnUpdate.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            //从编辑框中获得相应的属性值
            SQLiteDatabase dbWriter = dbOpenHelper.getReadableDatabase();
            ContentValues cv = new ContentValues();
            cv.put("remindTitle", taskEdit.getText().toString());
            cv.put("modified", 1);
            cv.put("modifyDate", longDateFormatter.format(System.
                currentTimeMillis()));
            cv.put("remindDate", longDateFormatter.format(newRemindDate.
                getTimeInMillis()));
            cv.put("remindText", remarkEdit.getText().toString());
            dbWriter.update("tb ToDoItem", cv, " id=?", new String[]
                {updateID}); //修改数据库中的数据
            getFragmentManager().popBackStack();
        }
    });
    return rootView;
}
}

```

4. 长按列表项的处理

长按列表项则可以删除该提醒项。因为删除操作是不可恢复的,所以删除之前弹出警告,提示用户确认删除操作,如图 12-10 所示。



图 12-10 删除列表中的提醒项

长按列表项的实现代码如代码段 12-17 所示。

代码段 12-17 设置长按列表项的响应

```
todoList.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
                                                                    //长按删除列表项  
    @Override  
    public boolean onItemClick(AdapterView<?>parent, View view, int  
        position, long id) {  
        HashMap<String,String> temp = (HashMap<String,String>)  
            listViewAdapter.getItem(position);  
        final String taskID=temp.get("_id");  
        String remindTitle=temp.get("remindTitle");  
        new AlertDialog.Builder(getActivity())  
            .setTitle("警告")  
            .setMessage("您要删除这条待办事项吗?"+"\\n\\n 待办事项标题:"  
                + remindTitle)  
            .setPositiveButton("删除", new DialogInterface.OnClickListener() {  
                public void onClick(DialogInterface arg0, int arg1) {  
                    SQLiteDatabase dbWriter = dbOpenHelper.getWritableDatabase();  
                    dbWriter.delete("tb_ToDoItem", "_id=?", new String[]{taskID});  
                                                                    //删除数据库中的数据  
                    dbWriter.close();  
                    getFragmentManager().beginTransaction()  
                        .replace(R.id.fragment_container, new RemindListFragment())
```

```
        .commit();  
    }  
    })  
    .setNegativeButton("取消", null)  
    .create()  
    .show();  
    return true;  
    }  
});
```

5. 添加新提醒项

添加新的提醒项通过 `AddNewFragment` 类实现,界面如图 12-11 所示。相关代码如代码段 12-18 所示。

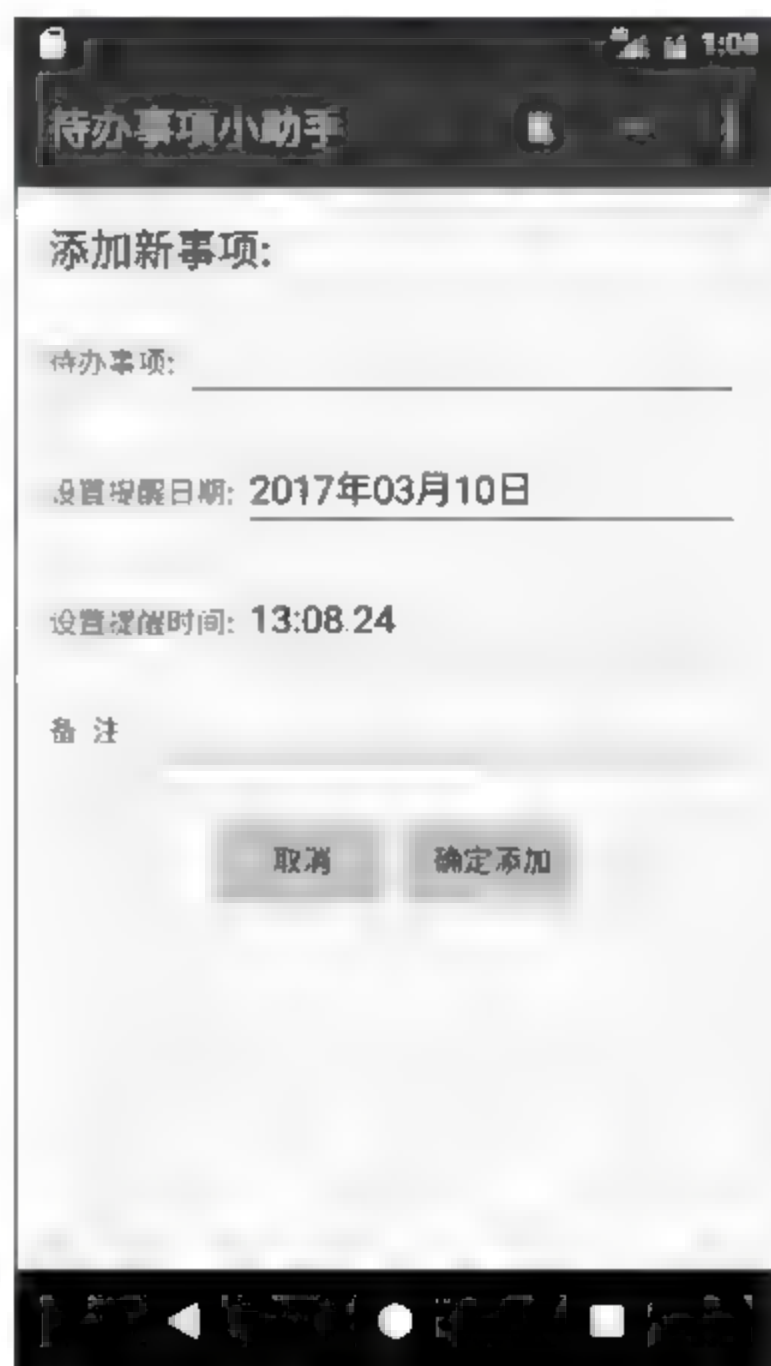


图 12-11 添加新提醒项

代码段 12-18 `AddNewFragment` 类的主要代码

```
//package 和 import 语句略  
public class AddNewFragment extends Fragment {  
    public AddNewFragment() {  
    }  
    private MyDBOpenHelper dbOpenHelper;
```

```
private Button btnAdd, btnCancel;
private EditText remindTitleEdit, dateEdit, timeEdit, remindTextEdit;
private Calendar createDate, remindDate;
private SimpleDateFormat dateFormatter, timeFormatter;
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_add, container, false);
    btnAdd = (Button) rootView.findViewById(R.id.btnAdd);
    btnCancel = (Button) rootView.findViewById(R.id.btnCancel);
    remindTitleEdit = (EditText) rootView.findViewById(R.id.etAddTask);
    dateEdit = (EditText) rootView.findViewById(R.id.etAddDate);
    timeEdit = (EditText) rootView.findViewById(R.id.etAddTime);
    remindTextEdit = (EditText) rootView.findViewById(R.id.etAddRemark);
    dbOpenHelper = new MyDBOpenHelper(getActivity().getApplicationContext());
    dateFormatter = new SimpleDateFormat("yyyy年MM月dd日");
    timeFormatter = new SimpleDateFormat("HH:mm:ss");
    createDate = Calendar.getInstance();
    createDate.setTimeInMillis(System.currentTimeMillis());
    remindDate = Calendar.getInstance();
    dateEdit.setText(dateFormatter.format(new Date(remindDate.
        getTimeInMillis())));
    timeEdit.setText(timeFormatter.format(new Date(remindDate.
        getTimeInMillis())));
    dateEdit.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            new DatePickerDialog(getActivity(), new DatePickerDialog.
                OnDateSetListener() {
                    @Override
                    public void onDateSet(DatePicker datePicker, int year, int
                        month, int day) {
                        remindDate.set(year, month, day);
                        dateEdit.setText(dateFormatter.format(new Date(remindDate.
                            getTimeInMillis())));
                    }
                }, remindDate.get(Calendar.YEAR), remindDate.get(Calendar.
                    MONTH), remindDate.get(Calendar.DAY_OF_MONTH))
                .show();
        }
    });
    timeEdit.setOnClickListener(new View.OnClickListener() {
        @Override
```

```

        public void onClick(View view) {
            new TimePickerDialog(getActivity(), new TimePickerDialog.
                OnTimeSetListener() {
                    @Override
                    public void onTimeSet(TimePicker timePicker, int
                        hourOfDay, int minute) {
                        remindDate.set(remindDate.get(Calendar.YEAR),
                            remindDate.get(Calendar.MONTH), remindDate.get
                                (Calendar.DAY_OF_MONTH), hourOfDay, minute);
                        timeEdit.setText(timeFormatter.format(new Date
                            (remindDate.getTimeInMillis())));
                    }
                }, remindDate.get(Calendar.HOUR_OF_DAY), remindDate.get
                    (Calendar.MINUTE), true)
                .show();
        }
    };

    btnCancel.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            getFragmentManager().beginTransaction()
                .replace(R.id.fragment_container, new RemindListFragment())
                .commit();
        }
    });

    btnAdd.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            //从编辑框中获得相应的属性值
            SimpleDateFormat longDateFormatter = new SimpleDateFormat
                ("yyyy-MM-dd HH:mm:ss");
            SQLiteDatabase dbWriter = dbOpenHelper.getWritableDatabase();
            ContentValues cv = new ContentValues();
            cv.put("remindTitle", remindTitleEdit.getText().toString());
            cv.put("createDate", longDateFormatter.format(new Date
                (System.currentTimeMillis())));
            cv.put("modifyDate", longDateFormatter.format(new Date
                (System.currentTimeMillis())));
            cv.put("remindDate", longDateFormatter.format(remindDate.
                getTimeInMillis()));
            cv.put("remindText", remindTextEdit.getText().toString());
            dbWriter.insert("tb_ToDoItem", null, cv); //向数据库添加数据
            dbWriter.close();
            //启动服务,定时推送 Notification 提醒:
            startTimeService(remindDate.getTimeInMillis() - System.

```

```
        currentTimeMillis(), remindTitleEdit.getText().toString(),  
        remindTextEdit.getText().toString());  
        //回到提醒项列表首页面:  
        getFragmentManager().beginTransaction()  
            .replace(R.id.fragment_container, new RemindListFragment())  
            .commit();  
    }  
});  
return rootView;  
}  
}
```

6. 列出今日提醒

点击操作栏的“今日提醒”图标,显示区只列出今日的提醒项。此功能通过 TodayListFragment 类实现,界面如图 12-12 所示。相关代码如代码段 12-19 所示。



图 12-12 定时推送的状态栏提醒

代码段 12-19 TodayListFragment 类的主要代码

```
//package 和 import 语句省略  
public class TodayListFragment extends Fragment {  
    private SQLiteDatabase dbRead;
```

```

private MyDBOpenHelper dbHelper;
private ListView ListTask;
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_today_list,
        container, false);
    ListTask = (ListView) rootView.findViewById(R.id.listTodayToDo);
    TextView tvToday = (TextView) rootView.findViewById(R.id.tvToday);
    SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy年MM月dd日");
    tvToday.setText(dateFormatter.format(new Date(System.currentTimeMillis())));
    dbHelper = new MyDBOpenHelper(getActivity().
        getApplicationContext());
    dbRead = dbHelper.getReadableDatabase();
    readToDoList();
    return rootView;
}
protected void readToDoList() {
    SimpleDateFormat dayFormatter = new SimpleDateFormat("yyyy-MM-dd");
    ArrayList taskList = new ArrayList<HashMap<String, String>>();
    Cursor result = dbRead.query("tb_ToDoItem", new String[]{"_id",
        "remindTitle", "createDate", "modified", "modifyDate",
        "remindText", "remindDate", "haveDo"},
        null, null, null, null, "createDate", null);
    while(result.moveToNext()) {
        if (result.getString(6).substring(0, 10).compareTo(
            dateFormatter.format(new Date(System.currentTimeMillis())))
            == 0) {
            HashMap<String, String> temp = new HashMap<String, String>();
            temp.put("_id", String.valueOf(result.getInt(0)));
            temp.put("remindTitle", result.getString(1));
            temp.put("createDate", "创建时间:" + result.getString(2));
            temp.put("modified", result.getInt(3) == 0 ? "未修改" : "已修改");
            temp.put("modifyDate", "最后修改时间:" + result.getString(4));
            temp.put("remindText", "备注:" + result.getString(5));
            temp.put("remindDate", "时间:" + result.getString(6));
            temp.put("haveDo", result.getInt(7) == 0 ? "该事项未处理" : "该事项
                已经处理");
            taskList.add(temp);
        }
    }
    final SimpleAdapter listViewAdapter =
        new SimpleAdapter(getActivity(), taskList, R.layout.today_
            list_item,

```

```
        new String[] {"remindDate", "remindTitle", "remindText",  
            "haveDo"},  
        new int[] {R.id.remind_listitem_remindDate, R.id.remind_  
            listitem_taskTitle,  
            R.id.remind_listitem_taskText, R.id.remind_listitem_haveDo} );  
        ListTask.setAdapter(listViewAdapter);    //将查询的结果显示到 ListView 控件中  
    }  
}
```

12.2.4 定时推送状态栏提醒

当添加一条新提醒项或修改提醒项的提醒时间后,要设置一个定时推送的状态栏提醒。这样,每一个提醒项预设的提醒时间到了之后,应用程序在状态栏就会推送 Notification 消息,提醒用户有需要处理的待办事项,如图 12-13 所示。

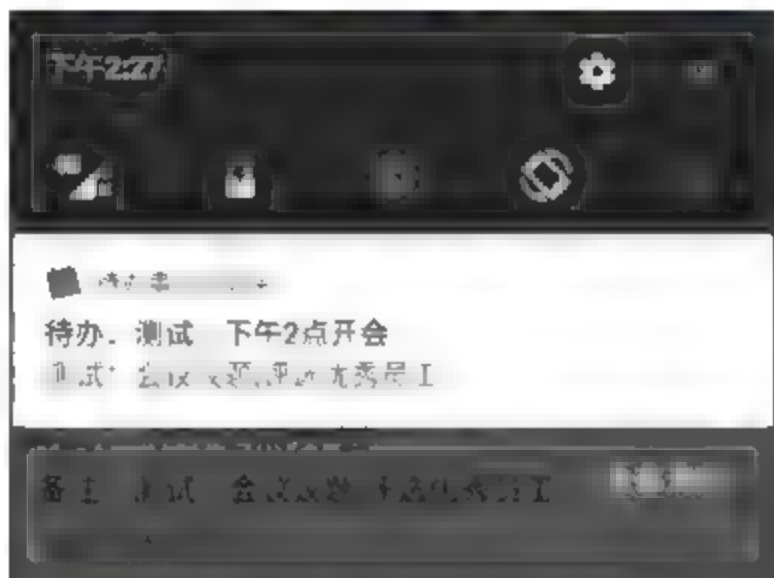


图 12-13 定时推送的状态栏提醒

定时推送 Notification 消息的功能通过启动服务来实现。首先要定义实现提醒定时推送的服务类 TimeService, 如代码段 12-20 所示。该服务类需要在 AndroidManifest.xml 文件中声明。

代码段 12-20 TimeService 类的主要代码

//package 和 import 语句省略

```
public class TimeService extends Service {  
    private Timer timer;  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        timer = new Timer(true);    //创建 Timer 对象  
    }  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        final String notificationTitle = intent.getStringExtra("title");
```

```

        final String notificationText = intent.getStringExtra("text");
        final int notificationID= intent.getIntExtra("notificationID",0);
        Long waitTime= intent.getLongExtra("time",0);
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                NotificationManager manager = (NotificationManager)
                    getSystemService(Context.NOTIFICATION_SERVICE);
                //获得通知管理器

                Notification.Builder myBuilder= new Notification.Builder
                    (TimeService.this);
                myBuilder.setSmallIcon(R.drawable.ic_warning)
                    .setContentTitle(getText(R.string.notification_title)
                        +notificationTitle) //定义通知的标题
                    .setContentText(notificationText) //定义通知的内容
                    .setDefaults(Notification.DEFAULT_SOUND)
                    //定义铃声
                    .setTicker(getText(R.string.notification_ticker));
                //定义提醒文字

                Notification notification=myBuilder.build();
                //创建通知,至少:minSdkVersion="16"

                manager.notify(notificationID, notification); //显示通知
            }
        }, waitTime);
        return super.onStartCommand(intent, flags, startId);
    }
}

```

当需要设置一个定时推送的状态栏提醒时,就启动这个服务,定时时间到后就会显示 Notification 消息。本例通过调用 `startTimeService()` 方法启动 TimeService 服务,该方法的定义如代码段 12-21 所示。`startTimeService()` 方法有 3 个参数,分别是用毫秒数表示的推送时间、Notification 的标题、Notification 的提示文字。

代码段 12-21 `startTimeService()` 方法的定义

```

private void startTimeService(Long time,String title,String text){
    int notificationID;
    SQLiteDatabase dbRead= (new MyDBOpenHelper(getActivity().
        getApplicationContext())).getReadableDatabase();
    Intent intent= new Intent(getActivity().getApplicationContext(),
        TimeService.class);
    Cursor result= dbRead.query("tb_notificationID",new String[]{"",
        notificationID"},null,null,null,null,null,null);
}

```

```

if (result.moveToFirst()) {
    notificationID= result.getInt(0);
    SQLiteDatabase dbWriter= (new MyDBOpenHelper(getActivity().
        getApplicationContext()).getWritableDatabase());
    ContentValues cv =new ContentValues();
    cv.put("notificationID", notificationID+ 1);
    dbWriter.update("tb_notificationID", cv,null, null);
    dbWriter.close();
}else {
    notificationID=0;           //没有获取数据库中的 notificationID 值,设为默认值 0
}
dbRead.close();
intent.putExtra("time", time);
intent.putExtra("title",title);
intent.putExtra("text",text);
intent.putExtra("notificationID",notificationID);    //传递参数
getActivity().startService(intent);                //启动 service
}

```

12.2.5 菜单设计

本例使用菜单实现界面切换、退出、查看版权信息的功能,菜单的显示效果如图 12 14 所示。设置了两个操作栏操作项,分别为“添加新事项”和“今日提醒”,如图 12 15 所示。

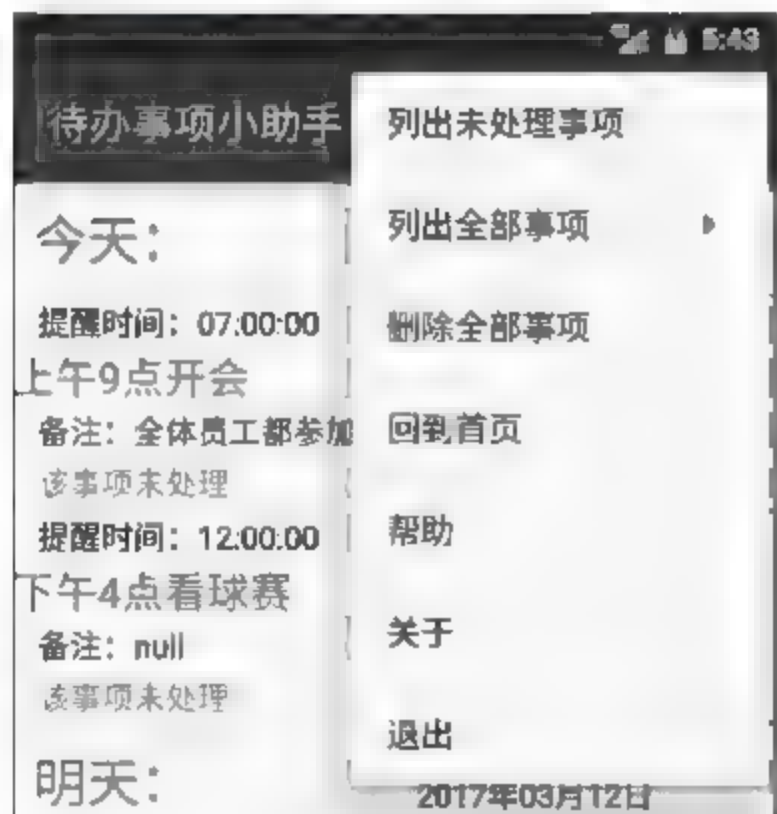


图 12-14 待办事项小助手的菜单



图 12-15 操作栏操作项

菜单采用 XML 方式实现。先在 res/menu 文件夹中新建 menu.xml 文件,在其中添加菜单项,相关代码如代码段 12-22 所示。其中前两个菜单项的 showAsAction 属性值分别是 always 和 ifRoom,将其设置为操作栏操作项。

代码段 12-22 定义菜单项

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <group android:id="@+id/group1">
        <item android:id="@+id/menu_add"
            android:title="添加新事项"
            android:icon="@mipmap/ic_add"
            app:showAsAction="always|withText"/>
        <item android:id="@+id/menu_today"
            android:title="今日提醒"
            android:icon="@mipmap/ic_remind"
            app:showAsAction="ifRoom"/>
        <item android:id="@+id/menu_undo"
            android:title="列出未处理事项"/>
        <item android:id="@+id/menu_list"
            android:title="列出全部事项">
            <menu>
                <item
                    android:id="@+id/menu_list_todo"
                    android:title="按待办时间排序"/>
                <item
                    android:id="@+id/menu_list_create"
                    android:title="按创建时间排序"/>
            </menu>
        </item>
        <item android:id="@+id/menu_clear"
            android:title="删除全部事项"/>
        <item android:id="@+id/menu_first"
            android:title="回到首页"/>
    </group>
    <group>
        <item android:id="@+id/menu_help"
            android:title="帮助"/>
        <item android:id="@+id/menu_about"
            android:title="关于"/>
        <item android:id="@+id/menu_exit"
            android:title="退出"/>
    </group>
</menu>
```

重写 MainActivity 中的 onCreateOptionsMenu() 方法,在界面中添加菜单。本例中

调用 `inflate()` 方法生成菜单, 该方法使用一个指定的 XML 资源填充菜单, 这里指定的是前一步骤创建的 `menu.xml` 文件。相关代码如代码段 12-23 所示。

代码段 12-23 添加菜单

```
public boolean onCreateOptionsMenu(Menu menu) {  
    MenuInflater inflater = getMenuInflater();           //获得 menu 容器  
    inflater.inflate(R.menu.menu, menu);                 //用 menu.xml 填充 menu 容器  
    return super.onCreateOptionsMenu(menu);  
}
```

重写 `onOptionsItemSelected(MenuItem item)` 方法, 实现各个菜单项的功能, 如代码段 12-24 所示。

代码段 12-24 实现菜单项的功能

```
public boolean onOptionsItemSelected(MenuItem item) {  
    switch(item.getItemId()) {  
        case R.id.menu_add:  
            getFragmentManager().beginTransaction()  
                .replace(R.id.fragment_container, new AddNewFragment())  
                .commit();  
            return true;  
        case R.id.menu_clear:                               //删除全部待办事项  
            showClearAll();  
        case R.id.menu_list_todo:                           //按待办时间顺序列出全部待办事项  
            getFragmentManager().beginTransaction()  
                .replace(R.id.fragment_container, new  
                    AllListByToDoTimeFragment())  
                .addToBackStack(null)  
                .commit();  
            return true;  
        case R.id.menu_list_create:                         //按创建时间顺序列出全部待办事项  
            getFragmentManager().beginTransaction()  
                .replace(R.id.fragment_container, new  
                    AllListByCreateTimeFragment())  
                .addToBackStack(null)  
                .commit();  
            return true;  
        case R.id.menu_today:                               //列出今日提醒  
            getFragmentManager().beginTransaction()  
                .replace(R.id.fragment_container, new TodayListFragment())  
                .addToBackStack(null)  
                .commit();  
            return true;  
        case R.id.menu_first:                               //回到首页
```

```
        getFragmentManager().beginTransaction()
            .replace(R.id.fragment_container, new RemindListFragment())
            .commit();
        return true;
    case R.id.menu_undo:           //列出未处理事项
        getFragmentManager().beginTransaction()
            .replace(R.id.fragment_container, new UndoListFragment())
            .commit();
        return true;
    case R.id.menu_help:           //帮助
        showHelp();
        return true;
    case R.id.menu_exit:           //退出
        showExit();
        return true;
    case R.id.menu_about:          //关于
        showAbout();
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

12.3 本章小结

本章主要介绍了两个 Android 综合应用程序的设计思路和实现方法,这些应用涉及了前几章学习过的界面组件、Fragment、启动服务、SQLite 数据库等。通过这些实例可以加深对基本知识的理解,提高综合应用能力。

习 题

1. 编写一个备忘录程序,实现备忘信息及提醒时间的输入、删除、修改和保存以及预定时间到达后的自动提醒。
2. 编写一个存款管理程序,用户将每笔存款的金额、存入银行的时间、存期以及支取金额、时间记录在一个数据库中,存期种类和利率如表 12-3 所示。要求每笔存款到期后要给用户一个 Notification 提醒;用户随时可查当前能支取的存款总额(定期存款随时可支取,但不到期的以活期计算利率);给用户提供参数设置界面,当银行的存款利率发生变化时,能及时设置新的存款利率。利率变化日之前存入的存款按旧利率计算,利率变化日之后存入的存款按新利率计算。

表 12-3 银行存款利率表

存 期	年利率/%	存 期	年利率/%
活期	0.35	两年	3.75
三个月	2.85	三年	4.25
六个月	3.05	五年	4.75
一年	3.25		

3. 编写一个个人记账软件,实现对支出和收入的记录。要求能够查询当前余额,按月查询和统计支出和收入情况,能够根据不同的类别查看自己的支出记录。

参考文献

- [1] 谷歌公司. Android 开发者指南. <http://developer.android.com/reference/>.
- [2] 百度百科. 智能手机操作系统. <https://baike.baidu.com/item/智能手机操作系统/5833789?fr=aladdin>.
- [3] 中国互联网络信息中心. <http://www.cnnic.net.cn/>.
- [4] 中文互联网数据资讯中心. <http://www.199it.com/>.
- [5] Statista Company. Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2017. <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>.
- [6] Deitel H M, Deitel P J. Java 语言程序设计大全. 袁晓靖, 等译. 北京: 机械工业出版社, 1997.
- [7] 范春梅, 张卫华. XML 基础教程. 北京: 人民邮电出版社, 2009.
- [8] 陈作聪, 苏静, 王龙. XML 实用教程. 北京: 机械工业出版社, 2014.
- [9] 马伟奇. 优化 Android Studio/Gradle 构建. (2015-6-15). <http://bbs.itheima.com/thread-204217-1-1.html>.
- [10] 高凯, 王俊社, 仇晶. Android 智能手机软件开发教程. 北京: 国防工业出版社, 2012.
- [11] 毋建军, 徐振东, 林瀚. Android 应用开发案例教程. 北京: 清华大学出版社, 2013.
- [12] 张思民. Android 应用程序设计. 北京: 清华大学出版社, 2013.
- [13] 吴亚峰, 于复兴, 杜化美. Android 应用案例开发大全. 北京: 人民邮电出版社, 2013.
- [14] Reto Meier. Android4 高级编程. 余建伟, 赵凯, 译. 北京: 清华大学出版社, 2013.
- [15] 余志龙, 陈昱勋, 郑名杰, 等. Google Android SDK 开发范例大全. 2 版. 北京: 人民邮电出版社, 2010.
- [16] 陈佳, 李树强. Android 移动开发. 北京: 人民邮电出版社, 2016.
- [17] 肚皮会唱歌. Activity 四种启动模式. (2012-8-23). <http://blog.csdn.net/shinay/article/details/7898492>.
- [18] nBlogs. Android 中 Application 类的用法. <http://www.cnblogs.com/renqingping/archive/2012/10/24/Application.html>.
- [19] 极客学院. <http://www.jikexueyuan.com/>.
- [20] anzhuo. AndroidRSS 阅读器源码. (2011-4-22). <http://www.apkbus.com/android-507-1-1.html>.
- [21] 害羞雏田. Android 中的 Handler 的具体用法. (2011-2-28). <http://txlong-onz.iteye.com/blog/934957>.
- [22] Android 开发者社区. <http://www.eoeandroid.com/forum.php>.
- [23] 安卓巴士. <http://www.apkbus.com/portal.php>.
- [24] CSDN 移动开发频道. <http://mobile.csdn.net/>.